Zur interdisziplinären Entwicklung sicherheitskritischer Assistenz- und Automationssysteme im Automobil

Jan Gačnik

Berichte aus dem DLR-Institut
für Verkehrssystemtechnik

Band 16

DLR Deutsches Zentrum
für Luft- und Raumfahrt

Zur interdisziplinären Entwicklung sicherheitskritischer Assistenz- und Automationssysteme im Automobil

Von der Fakultät für Maschinenbau
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung der Würde

eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation

von: Dipl.-Wirtsch.-Ing Jan Gačnik

aus (Geburtsort): Hannover

eingereicht am: 09. Juni 2011

mündliche Prüfung am: 12. Oktober 2011

Referenten: Prof. Dr.-Ing. Karsten Lemmer, Prof. Dr. Bernhard Josko

2012

Berichte aus dem DLR-Institut für Verkehrssystemtechnik

Band 16

Zur interdisziplinären Entwicklung sicherheitskritischer Assistenz- und Automationssysteme im Automobil

Jan Gačnik

Herausgeber:
Deutsches Zentrum für Luft- und Raumfahrt e.V.
Institut für Verkehrssystemtechnik
Lilienthalplatz 7, 38108 Braunschweig

ISSN 1866-721X

DLR-TS 1.16

Braunschweig, im Februar 2012

Institutsdirektor:
Prof. Dr.-Ing. Karsten Lemmer

Verfasser:
Jan Gačnik

Vorwort des Herausgebers

Liebe Leserinnen und Leser,

In Ihren Händen halten Sie einen Band unserer Buchreihe „Berichte aus dem DLR-Institut für Verkehrssystemtechnik". In dieser Reihe veröffentlichen wir spannende, wissenschaftliche Themen aus dem Institut für Verkehrssystemtechnik des Deutschen Zentrums für Luft- und Raumfahrt e.V. (DLR) und aus seinem Umfeld. Einen Teil der Auflage stellen wir Bibliotheken und Fachbibliotheken für ihren Buchbestand zur Verfügung. Herausragende wissenschaftliche Arbeiten und Dissertationen finden hier ebenso Platz wie Projektberichte und Beiträge zu Tagungen in unserem Hause von verschiedenen Referenten aus Wirtschaft, Wissenschaft und Politik.

Mit dieser Veröffentlichungsreihe verfolgen wir das Ziel, einen weiteren Zugang zu wissenschaftlichen Arbeiten und Ergebnissen zu ermöglichen. Wir nutzen die Reihe auch als praktische Nachwuchsförderung durch die Publikation der wissenschaftlichen Ergebnisse von Dissertationen unserer Mitarbeiter und auch externer Doktoranden. Veröffentlichungen sind wichtige Meilensteine auf dem akademischen Berufsweg. Mit der Reihe „Berichte aus dem DLR-Institut für Verkehrssystemtechnik" erweitern wir das Spektrum der möglichen Publikationen um einen Baustein. Darüber hinaus verstehen wir die Kommunikation unserer Forschungsthemen als Beitrag zur nationalen und internationalen Forschungslandschaft auf den Gebieten Automotive, Bahnsysteme und Verkehrsmanagement.

Im vorliegenden Band wird eine neue Methode vorgestellt, welche die Entwickler von Assistenz- und Automationssystemen bei der Bewältigung komplexer Entwicklungsprozesse über die Grenzen der eigenen Disziplin hinaus unterstützt. Kern der Methode ist die Formalisierung von Domänenwissen, welche die Komplexität von Produkten und Prozessen handhabbar macht und Inkonsistenzen vorbeugt. So wird den Entwicklern problemorientiert und zielgerichtet relevantes Fachwissen anderer Fachdisziplinen aufbereitet und zur Verfügung gestellt. Damit können Entwurfsalternativen sowie der Einfluss von Änderungen frühzeitig im Gesamtkontext bewertet werden. Mit dieser Methode zur effizienten Verknüpfung interdisziplinären Wissens liefert die vorliegende Arbeit so einen Beitrag für die Verbesserung der Entwicklungszeit und der Qualität zukünftiger Assistenz- und Automationssysteme.

Prof. Dr.-Ing. Karsten Lemmer

Vorwort des Autors

Die vorliegende Arbeit entstand während meiner wissenschaftlichen Tätigkeit beim Institut für Verkehrssystemtechnik (TS) am Deutschen Zentrum für Luft- und Raumfahrt (DLR) in Braunschweig.

Ich möchte meinem Doktorvater Herrn Prof. Dr.-Ing. Karsten Lemmer für die zielorientierte Betreuung und Förderung meiner Arbeit und meiner Person herzlich danken. Ebenso danke ich Herrn Prof. Dr. Bernhard Josko für die Übernahme des Koreferats und die konstruktiven, detailreichen Diskussionen zu meiner Arbeit. Weiterhin freue ich mich über die Übernahme des Vorsitzes der Prüfungskommission durch Herrn Prof. Dr.-Ing. Dr. h.c. mult. Eckehard Schnieder.

Darüber hinaus möchte ich mich bei Herrn PD Dr. Frank Köster für die intensive Unterstützung bei der Erstellung meiner Arbeit und für die vielen konstruktiven und anregenden Diskussionen bedanken. Dasselbe gilt für Herrn Dipl.-Inform. Henning Jost von der Universität Oldenburg, der mich fachlich wie persönlich stark unterstützt hat.

Weiterer Dank für Korrekturen und Diskussionen zu meiner Arbeit gilt meinen ehemaligen Kollegen im DLR. Besonders hervorheben möchte ich dabei Frau Dipl.-Ing. Silke Köhler und Herrn Dipl.-Inform. Marco Hannibal.

Meiner Familie danke ich für die jahrelange persönliche Unterstützung während meines Studiums und der folgenden wissenschaftlichen Arbeit. Weiterhin möchte ich mich ganz herzlich bei meiner Freundin Frederike Feldmann bedanken, die mich immer wieder in den richtigen Momenten motiviert und unterstützt hat.

Dipl.-Wirtsch.-Ing. Jan Gacnik

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzfassung

Assistenz- und Automationssysteme im Automobil zeichnen sich verstärkt durch eine große Komplexität und hohen Vernetzungsgrad aus. Weiterhin existiert eine Vielzahl von Anforderungen anderer Fachdisziplinen an die Funktionsentwicklung, wie beispielsweise in den Bereichen der funktionalen Sicherheit und der Fahrer-Fahrzeug-Interaktion. Somit ergibt sich für die Entwickler nicht nur eine gestiegene Komplexität der Assistenzsysteme, sondern auch der einzuhaltenden Entwicklungsprozesse.

Die vorliegende Arbeit liefert eine neue Methode, die Entwickler bei der Bewältigung dieser Komplexität unterstützt. Dabei wird ihnen problemorientiert und zielgerichtet relevantes Wissen anderer Fachdisziplinen aufbereitet und zur Verfügung gestellt. Somit können Entwurfsalternativen früher als bisher im Kontext der Gesamtentwicklung bewertet werden, was sich positiv auf Produktqualität und Entwicklungszeit auswirkt.

Eine wichtige Basis dieser Methode ist die Formalisierung des Entwicklungsprozesses. Dazu wird ein iteratives und rekursives Vorgehensmodell definiert, welches verschiedene Disziplinen und ihr Domänenwissen integriert. Die Formalisierung ist mit einer großen Offenheit gestaltet, um das formalisierte Domänenwissen langfristig und flexibel in verschiedenen Projekten wiederverwenden zu können. Zur Formalisierung von Domänenwissen wird die Web Ontology Language (OWL) verwendet, eine international standardisierte Sprachfamilie zur maschinenlesbaren Wissensrepräsentation.

In einem weiteren Schritt beschreibt diese Arbeit, wie auf Basis des formalisierten Domänenwissens die interdisziplinären Entwicklungsaktivitäten effizient verknüpft werden können. Dazu wird die Analysierbarkeit des formalisierten Wissens für automatische Schlussfolgerungen auf der vorhandenen Wissensbasis genutzt. So werden insbesondere Einflussanalysen möglich, um über die eigene Domäne hinaus Änderungen bezüglich des Gesamtentwicklungsprozesses zu bewerten.

Im Rahmen dieser Arbeit wurde eine prototypische Werkzeugkette implementiert, die die beschriebene Methode umsetzt und generische Analyseschritte auf Basis der OWL implementiert. Durch die Werkzeugintegration kann der Nutzen der Wissensaufbereitung für einen Funktionsentwickler aufgezeigt und Hinweise bezüglich der technischen Skalierbarkeit der Methode gegeben werden.

Als Anwendungsbeispiel im Rahmen der prototypischen Implementierung dient die angedeutete Weiterentwicklung eines radarbasierten Abstandsregeltempomaten zu einem Notbremssystem. Dazu werden funktionale Anforderungen mit einem Spezifikationswerkzeug modelliert und annotiert. Eine Einflussanalyse über das zuvor formalisierte Domänenwissen liefert daraufhin ein maßgeschneidertes Vorgehensmodell. Dabei wird insbesondere der Einfluss der funktionalen Sicherheit auf die Funktionsentwicklung beleuchtet, indem aus einem formalisierten Wissensmodell des Standards ISO 26262 notwendige Anforderungen und Methoden für den Gesamtentwicklungsprozess abgeleitet werden.

Abstract

Automotive assistance and automation systems increasingly present high levels of complexity and connectivity. In addition, more and more specific requirements in the disciplines of functional safety and driver-vehicle interaction have to be considered during functional development. Thus, developers have to cope with increasing levels of product complexity, but also complexity of the development process.

This thesis presents a new method which supports developers in handling these dimensions of complexity. Therefore relevant knowledge of other disciplines is presented to them in a solution-oriented way. Thus, design alternatives can be assessed much earlier in the overall development process which has a positive impact on product quality and development cycles.

An important base in this method is the formalization of the development process. Therefore an iterative and recursive process model is defined which integrates different development disciplines and their domain knowledge. The formalization is designed in a very open way to enable long-term and flexible reuse in several development projects. For formalizing domain knowledge the Web Ontology Language (OWL) is used which is an internationally standardized family of languages for machine-readable knowledge representation.

The next aspect in this thesis involves efficient use of formalized domain knowledge to link different interdisciplinary development activities. Therefore the analyzability of formalized knowledge is used for automated reasoning on the assembled knowledge base. This especially introduces an impact analysis to assess major cross-cutting changes to the product in the context of the overall development process.

For this thesis a prototype toolchain is developed which implements the described method and provides generic analysis steps on top of OWL. Through the technical implementation the benefit of formalizing domain knowledge for functional developers can be demonstrated. Furthermore, hints can be derived regarding technical scalability of the method.

The extension of a radar-based adaptive cruise control system towards an emergency braking system is sketched as an example for the prototype toolchain. Functional requirements are modelled and annotated using a requirements specification tool. Impact analysis on the formalized domain knowledge provides a custom tailored process model. In this example, especially the impact of functional safety on the functional development is focussed, involving a formalized model of requirements and methods from ISO 26262.

1 Motivation und Forschungsfragen

1.1 Entwicklung von Assistenz- und Automationssystemen

In Bezug auf das Verkehrsmittel Automobil sind die Schonung von *Ressourcen* und die Verbesserung der *Sicherheit* des Verkehrs zwei herausragende Forschungsziele. Hierbei ist die Schonung von Ressourcen im Allgemeinen recht weit gefasst und schließt z.B. den Schutz der Umwelt ebenso ein, wie die Gewinnung von Lebenszeit durch Zeitersparnis während zurückgelegter Wegstrecken. Bezogen auf die Schonung der Umwelt sind innerhalb der Europäischen Union fordernde Ziele vorgegeben. So ist etwa bis zum Jahre 2012 das Emissionsziel von 120 Gramm Kohlendioxid pro Kilometer zu erreichen [MED07a][Bun06]. Eine verbesserte Assistenz und ein höherer Grad an Automatisierung bei der Fahrzeugführung können wesentlich zur Erreichung der gesetzten Emissionsziele beitragen. Weiterhin kann dieses auch helfen, Staus zu vermeiden [MED07b], und so insbesondere eine effektive Verkürzung der Reisezeit für Autofahrer zu erzielen. Darüber hinaus kann die Automatisierung einen wesentlichen Beitrag zu sinkenden Unfallzahlen und somit zur Vision des unfallfreien Fahrens leisten [ZVE09].

Deshalb liegt eine wichtige Aktivität zur Erreichung der genannten Forschungsziele in der Entwicklung von *Assistenz- und Automationssystemen* (AAS). Hierbei umfassen die AAS der vorliegenden Arbeit sowohl die Menge der *Advanced Driver Assistance Systems* (ADAS) [Eur06b], als auch die Menge der *in-Vehicle Information Systems* (IVIS) [Eur06b]. Weiterhin fallen darunter auch solche Systeme, die über den unmittelbaren Fahrzeugkontext hinaus unterstützen, wie ein *Travel Assistance System* (TAS) [GHHK08]. Diese Systeme integrieren sich mehr und mehr in verteilte Systemlandschaften, wozu *Car-2-X Communication Systems* (C2X) [CAR07] einen wesentlichen Beitrag leisten.

In engem Zusammenhang mit einer komplexen Automatisierung lassen sich Trends bei der Entwicklung innovativer eingebetteter Systeme im Automobil beobachten (siehe dazu auch [ZVE09]). Diese betreffen das eigentliche *Produkt* ebenso wie die *Prozesse*, die zur Produktentwicklung eingesetzt werden. So ist etwa eine *Vernetzung und Flexibilisierung* von Produktkomponenten und Prozessaktivitäten notwendig, um kooperative AAS realisieren zu können. Eine *Verteilung und Dezentralisierung* von Entwicklungsaktivitäten ermöglicht kürzer werdende Produktlebenszyklen und umfasst die Einrichtung von *Lieferketten*. Für eine verteilte Entwicklung ist *Interoperabilität* zentral. Diese kann bei eher statischen und geschlossenen Anwendungen durch eine *Standardisierung* und damit verbundener *Konvergenz* umgesetzt werden. Bei offenen und flexiblen Anwendungen, wo eine Standardisierung nur schwer realisierbar ist, dominieren die Trends zur *Serviceorientierung* und *Integration*.

Im Zusammenhang mit einer starken Spezialisierung und Dezentralisierung, insbesondere auch durch Lieferketten, erhöht sich die Anzahl der an der Entwicklung beteiligten Firmen und Disziplinen.

Die genannten Trends führen zu einer steigenden Komplexität bei der Entwicklung sicherheitskritischer AAS im Automobil, die von Entwicklern bewältigt werden muss. Dieses betrifft die Prozesskomplexität ebenso wie die Produktkomplexität.

- *Produktkomplexität*: Einige Forschungsprojekte treiben AAS bis zur vollständig autonomen Fahrzeugführung in urbanen Umgebungen, womit sehr komplexe, sicherheitskritische AAS entstehen. Hierzu zählt etwa das Projekt *Stadtpilot* [WSM10, RGW+11]. Hinzu kommt auch im Beispiel Stadtpilot die Integration von AAS im Fahrzeug mit Verkehrsinfrastrukturdiensten zu einer Anwendungslandschaft [NFG+11, LICPG+11].

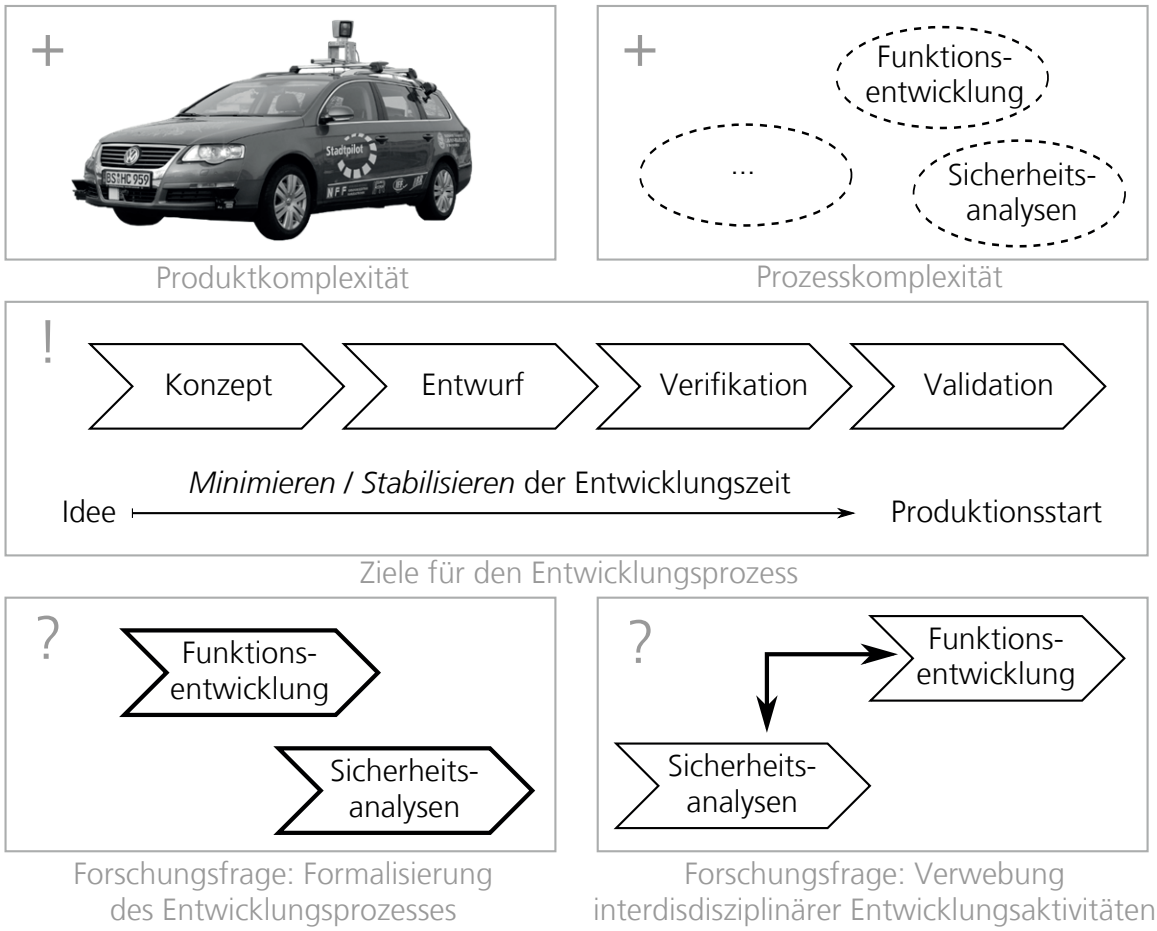

Abbildung 1-1: Trotz steigender Prozess- und Produktkomplexität soll eine kurze und stabile (planbare) Entwicklungszeit sichergestellt werden.

- *Prozesskomplexität*: Durch kürzere Produktlebenszyklen bei steigender Produktkomplexität wird eine stärkere Parallelisierung der Arbeiten notwendig, die durch eine steigende Prozessorientierung begleitet wird. Hinzu kommen neue juristisch bedingte Randbedingungen bei der Entwicklung, die sichergestellt werden müssen. So wird beispielsweise die Konformität zu Sicherheitsstandards [ISO09, IEC03b] wichtig.

Insgesamt wird ein großer Anteil Komplexität von der Produkt- auf die Prozessebene gehoben und muss in diesem Rahmen gelöst werden. Dabei erlangt die Planung und Kontrolle eines solchen disziplinübergreifenden Entwicklungsprozesses wesentliche Bedeutung, um zum Produktionsbeginn die gewünschte Qualität des Produktes zu erreichen (siehe Abbildung 1-1[1]). Dabei sind die *Verkürzung* und *Stabilisierung* (Planbarkeit) der Entwicklungszeit zentral. Die Prozessorientierung spielt weiterhin bei der juristischen Absicherung des Produktes eine wesentliche Rolle. Es existieren Normen und Standards, die sicherstellen sollen, dass juristisch und technisch nach dem Stand der Technik entwickelt wird. Entsprechende Prozesse unterstützen dabei die korrekte und vollständige Umsetzung solcher Normen und Standards.

Diese Arbeit leistet einen Beitrag zur Unterstützung der Entwickler bei steigender Prozess- und Produktkomplexität. Dabei greift sie Forschungsfragen auf, die sich aus den übergeordneten Forschungszielen *Sicherheit* und *Ressourcenschonung* ableiten lassen und fokussiert dabei die funktionale Softwareentwicklung im Rahmen der AAS Systementwicklung. In diesem Bereich bestimmen neben der Prozesskomplexität und Produktkomplexität eine Reihe weiterer Einflussfaktoren einen Projekterfolg. So kann sich etwa die Verwendung eines Vorgehensmodells vor allem positiv auf die Einhaltung der Projektlaufzeit bemerkbar machen [BEF+10]. Dieses ist

[1] Quelle Abbildung *Stadtpilot*: [RGW+11]

ein wesentlicher Erfolgsfaktor. Wichtig für den Projekterfolg und die Arbeit von Entwicklern ist weiterhin die Unterstützung und Einbindung von Führungskräften. Dieser Aspekt ist nach [EK08] zusammen mit zu vielen Änderungen von Produktanforderungen der wichtigste Grund (jeweils 33%) für scheiternde Softwareentwicklungsprojekte. Betrachtungen nach [BEF+10] ergeben weiterhin, dass vor allem ein früher Umgang mit sich ändernden Produktanforderungen für ein Projekt erfolgversprechend ist. Wesentliche Probleme bei der Spezifikation von Anforderungen liegen dabei in ihrer *Konsistenz*, *Korrektheit* und *Vollständigkeit*. Dieses sind Probleme, die insbesondere durch den interdisziplinären Charakter der Entwicklung von AAS noch verstärkt werden.

1.2 Forschungsfragen

Die Forschungsfragen dieser Arbeit lassen sich in die *Nationale Roadmap Embedded Systems* [ZVE09] einbetten, insbesondere in den Forschungsschwerpunkt *Virtual Engineering*. In diesem Forschungsschwerpunkt definiert die Roadmap einen langfristigen Forschungsbedarf im Bereich des *Multi-Domain Engineering* für einen Serieneinsatz zwischen 2020 und 2025. Dabei geht es insbesondere um virtuelle Entwicklung und Modellinteroperabilität zwischen verschiedenen Ingenieursdisziplinen. Aus dem *Multi-Domain Engineering* lassen sich somit auch die Forschungsfragen dieser Arbeit motivieren: die *Formalisierung des Entwicklungsprozesses* und die *Verwebung interdisziplinärer Entwicklungsaktivitäten*. Sie zielen auf eine Stabilisierung und Minimierung der Entwicklungszeit ab (siehe auch Abbildung 1-1).

Die *Formalisierung des Entwicklungsprozesses* ermöglicht Effizienzsteigerungen durch die Bereitstellung werkzeuggestützter Analysen und Prozessplanung. Auf der Ebene der Prozesse werden die verschiedenen Entwicklungsdisziplinen *makroskopisch* verknüpft. Dieses bildet außerdem die Grundlage für die *mikroskopische* Verbindung, die *Verwebung interdisziplinärer Entwicklungsaktivitäten*. Diese integriert Entwicklungsaktivitäten bezüglich der Dimensionen Methoden und Wissen mit dem Ziel, einzelne Entwickler durch den Gesamtentwicklungsprozess individuell zu unterstützen.

Die Bedeutung dieser abstrakten Problemstellung in der Praxis lässt sich an einem Beispiel erläutern: So muss ein Funktionsentwickler etwa damit umgehen, dass durch die Standardisierung der ISO 26262 als Norm für funktionale Sicherheit sein System gegebenenfalls fast 1000 zusätzliche Anforderungen umsetzen muss. Besonders kompliziert wird es für ihn dadurch, dass die Nichtbeachtung dieser Anforderungen nicht unmittelbar zu einem leicht zu testenden Fehlverhalten führt, sondern vor allem zu möglichen späteren juristischen Problemen. Dieses wirft für ihn konkrete Fragen auf: „Wurde bei der Entwicklung alles beachtet? Wie kritisch ist ein Ausfall des Systems, wenn der Fahrer nicht über den Ausfall informiert wird?“

Bezogen auf das vorliegende Beispiel zielt eine *Formalisierung des Entwicklungsprozesses* vor allem auf eine Werkzeugunterstützung des Entwicklers zur Bewältigung der Komplexität ab (z.B. wiederverwendbare Modelle der etwa 1000 Anforderungen). Die *Verwebung interdisziplinärer Entwicklungsaktivitäten* soll eine zielgerichtete Führung durch den Entwicklungsprozess ermöglichen, indem frühzeitig Wissen aus anderen Disziplinen bedarfsgerecht zur Verfügung gestellt werden (z.B. Bereitstellung der durch einzelne Entwickler umzusetzenden Sicherheitsanforderungen).

In Ergänzung zum (klassischen) *Systems Engineering* wird in dieser Arbeit vor allem auf die Integration nichttechnischer Randbedingungen und Einflüsse auf die Entwicklung eingegangen (wie etwa funktionale Sicherheit). Diese finden in die Gesamtentwicklung vor allem auf Ebene der Produktanforderungen Einzug, und nicht als unmittelbar zu integrierende Systemkomponenten.

1.2.1 Formalisierung des Entwicklungsprozesses

Die *Formalisierung des Entwicklungsprozesses* motiviert sich vor allem aus dem Ziel, mit der gestiegenen Entwicklungskomplexität effizient umzugehen. Dieses umfasst eine zeitliche Verkürzung und temporale Stabilität (Planbarkeit) des Gesamtentwicklungsprozesses. Ein formalisierter Entwicklungsprozess erfüllt dabei die Funktion, eine systematische und werkzeuggestützte Analyse von Entwurfsalternativen in der Planung und Kontrolle des Prozesses möglich zu machen. Dabei handelt es sich insbesondere um Analysen bezüglich Konsistenz, Korrektheit und Vollständigkeit.

Bei der Formalisierung ist der Aspekt der *Offenheit* sehr wichtig, um Modelle langfristig zu nutzen und wiederzuverwenden. Das bedeutet, dass bei der Formalisierung alle Fakten *explizit* formuliert werden. Dieses ist vor allem deshalb wichtig, da bspw. ein Modell der Anwendungsdomäne AAS in der Praxis nie vollständig sein kann. Die Anwendungsdomäne unterliegt einem stetigen Wandel.

Ein wichtiges Kriterium ist, ob eine solche *Formalisierung* mehr *Flexibilität* während des Entwurfs bedeuten kann. Dieses kann erfolgen, indem der Entwurfsraum bei der Prozessplanung auf sinnvolle Bereiche verkleinert wird. Somit kann die Entscheidungsfindung vereinfacht werden, was neue Freiräume schafft. Dazu ist eine Formalisierung von Prozess- und Produktmodellen notwendig. Denn sie ermöglicht eine problembezogene Verknüpfung zwischen Produkt und Prozess, die die Grundlage für eine Führung durch den Entwurf stellt.

1.2.2 Verwebung interdisziplinärer Entwicklungsaktivitäten

Die *Verwebung interdisziplinärer Entwicklungsaktivitäten* soll Entwickler zielgerichtet bei der Bewältigung der Entwicklungskomplexität unterstützen. Somit meint Verwebung in diesem Zusammenhang die konkrete Verknüpfung der Entwicklungsdisziplinen bezüglich der Dimensionen Wissen und Methoden. Dabei wird der formalisierte Entwicklungsprozess als Basis genutzt.

Auch für die Verwebung ist die *Offenheit* besonders wichtig. Denn gerade für einzelne Entwickler ist es nicht mehr möglich, alles über die gesamte Entwicklung eines AAS zu wissen. Somit ist es wichtig, dass konkrete Entwurfsentscheidungen auf explizit vorliegendem Wissen erfolgen, nicht auf impliziten Annahmen.

Zudem kommt der *Führung* der Entwickler und ihrer Aktivitäten durch den Entwicklungsprozess eine wichtige Rolle zu. Es soll sichergestellt werden können, dass eine individuelle Entwurfsentscheidung sich positiv auf den Gesamtentwurf auswirkt. Ferner muss ein einzelner Entwickler überhaupt in die Lage versetzt werden können, seine eigenen Entwurfsentscheidungen und deren Auswirkung im Kontext einer interdisziplinären Entwicklung reflektieren und bewerten zu können. Hierzu muss ihm externes Wissen zielgerichtet zur Verfügung gestellt werden.

1.3 Einbettung der vorliegenden Arbeit

Die vorliegende Arbeit greift die beiden Forschungsfragen *Formalisierung des Entwicklungsprozesses* und *Verwebung interdisziplinärer Entwicklungsaktivitäten* auf und leistet damit einen unmittelbaren Beitrag in den zwei Projekten DeSCAS (siehe Abschnitt 1.3.1) und CESAR (siehe Abschnitt 1.3.2).

1.3.1 Virtuelles Institut DeSCAS

Das virtuelle Institut *Design of Safety-Critical Automotive Systems* (DeSCAS) verbindet seit März 2007 das Institut für Verkehrssystemtechnik im Deutschen Zentrum für Luft- und Raumfahrt, das Forschungszentrum Sicherheitskritische Systeme der Carl von Ossietzky Universität Oldenburg und das Institut für Verkehrssicherheit und Automatisierungstechnik der Technischen Universität Braunschweig. DeSCAS wird von der Helmholtz Gemeinschaft gefördert und hat das Ziel, einen sicherheitsorientierten und gleichzeitig menschzentrierten Entwicklungsprozess zu definieren. Als wichtigster Teil steht bei DeSCAS [GJBK08] die *Formalisierung* und *Verwebung* dieser interdisziplinären Entwicklungsaktivitäten im Vordergrund. DeSCAS behandelt als Anwendungsbeispiel solche AAS, welche die longitudinale Bewegung eines Automobils beeinflussen. Dazu gehört etwa ein *Adaptive Cruise Control* (ACC) [ISO02b] System. Neben dem Forschungsauftrag ist mit DeSCAS die Qualifizierung wissenschaftlichen Nachwuchses verbunden.

1.3.2 Europäische Initiative CESAR

Seit März 2009 existiert das europäische Forschungsprojekt *Cost-Efficient methods and processes for SAfety Relevant embedded systems* (CESAR), in dessen Rahmen insgesamt 55 Partner zusammenarbeiten. Dabei gilt es, Methoden und Werkzeuge für industrielle Anwendungen zu entwickeln. Diese sollen die Entwicklung hochzuverlässiger eingebetteter Systeme ermöglichen, welche den gestiegenen gesellschaftlichen Bedarf an Mobilität decken. Das Projekt teilt sich in mehrere *Subprojects* (SP) auf. Dabei wird eine Referenztechnologieplattform (SP1) entwickelt, die durch Innovationen im *Requirements Engineering* (SP2) und *Component Based Design* (SP3) unterstützt wird. Diese Plattform wird in Domänen Automobil (SP5), Luftfahrt (SP6) und Bahn/Automatisierungstechnik (SP7) angewendet und evaluiert. Der Innovationsbedarf im Rahmen des CESAR SP2 wurde durch die Anwendungsdomänen identifiziert. Dabei wurde industrieller Bedarf an einer Prozessformalisierung formuliert, der durch den Stand der Technik [Pai10b] und Wissenschaft [Pai10a] noch nicht abgedeckt ist.

In diesem Zusammenhang dient die Prozessformalisierung vor allem dem vereinfachten Konformitätsnachweis bezüglich Normen und Standards der jeweiligen Anwendungsdomäne und der automatisierten Erzeugung von Dokumentation für die Zulassung. Aus der vorliegenden Arbeit wurde dabei die Prozessformalisierung in das Projekt übernommen, die für die Planung des Zertifizierungsprozesses genutzt wird. Vor allem für die Projektplanung werden neben der *Formalisierung* Methoden zur *Verwebung* von Entwicklungsaktivitäten aufgegriffen.

1.4 Ergebnisse der vorliegenden Arbeit

Diese Arbeit beschreibt eine neuartige Methode, die Entwickler bei der Bewältigung von Produkt- und Prozesskomplexität unterstützt. Dabei werden einem Entwickler externes Wissen und Randbedingungen (z.B. Anforderungen aus Normen und Standards) durch eine Werkzeugunterstützung maßgeschneidert zur Verfügung gestellt. Somit reduziert sich der Aufwand der Informationsbeschaffung und Abstimmungsbedarf bei verteilten Entwicklungsprojekten, wodurch sich die Planbarkeit und die Durchlaufzeit eines Projektes verbessert. Darüber hinaus erhöht sich die Qualität des entwickelten Produkts, da Entwicklern frühzeitig eine vollständigere Informations- und Entscheidungsbasis vorliegt. Neben einer erhöhten Kundenzufriedenheit können so auch Kosten hinsichtlich zu erwartender Reklamationen reduziert werden.

Der Aufwand für die Wissensaufbereitung für die Entwickler kann durch den ausgeprägten generischen Charakter schnell amortisiert werden, so können etwa Normen und Standards in vielen Entwicklungsprojekten wiederverwendet werden.

Besonders bemerkenswert und neu ist die Verbindung eines hohen *Formalisierungsgrades* zusammen mit einer hohen *Offenheit* der vorliegenden Methode durch den Einsatz geeigneter Beschreibungsmittel und Werkzeuge. Dabei ermöglicht der hohe Formalisierungsgrad gezielte Werkzeugunterstützung des Entwicklers, sowie Analysemöglichkeiten (z.B. in Richtung Vollständigkeit). Die Offenheit impliziert einen hohen Grad an Wiederverwendbarkeit von modelliertem Wissen, sowie dessen flexiblen und nachhaltigen Einsatz in heterogenen Werkzeuglandschaften.

Somit liefert diese Arbeit Vorschläge für die Beantwortung der Forschungsfragen in Abschnitt 1.2. Dabei wird ein formales Vorgehensmodell beschrieben, welches eine wissensbasierte Modellierung der Anwendungsdomäne beinhaltet. Weiterhin wird eine Methode dargestellt, die auf Basis des im Vorgehensmodell formalisierten Domänenwissens eine offene und flexible Verwebung interdisziplinärer Entwicklungsaktivitäten erlaubt. Dabei kann die formale Basis zur Definition von Analysen genutzt werden, die die Auswirkung von Entwurfsentscheidungen im interdisziplinären Kontext bewertbar machen. Darüber hinaus wird ein Konzept erarbeitet, welches eine technische Nutzung des formalisierten Wissens ermöglicht.

Die beschriebenen konzeptuellen Ergebnisse wurden prototypisch in einer Werkzeugkette umgesetzt. Die Entwicklung eines sicherheitskritischen AAS wird aus Sicht der Funktionsentwicklung skizziert, dabei werden notwendige Sicherheitsmaßnahmen und Prozesse aus einem wissensbasierten Modell der ISO DIS 26262 [ISO09] abgeleitet. Dieser Standard ist zum heutigen Zeitpunkt der wichtigste Standard für funktionale Sicherheit von AAS im Automobil.

1.5 Aufbau der Arbeit

Ein Gesamtüberblick über die Struktur der Arbeit ist in Abbildung 1-2[2] grafisch dargestellt, und wird im Weiteren erläutert. Dabei wird die Ebene des *projektbezogenen Wissens* (spezifisches Wissen konkreter Produktentwicklungen) genutzt, um die Forschungsfragen und Ergebnisse dieser Arbeit praktisch zu motivieren und zu erklären. Auf der Ebene des *Domänenwissens* liegen zwar anwendungsunabhängige, jedoch domänenspezifische Beschreibungen vor. Der größte Teil dieser Arbeit hingegen versucht, generisch nutzbares Wissen zu *Beschreibungsmitteln, Methoden und Werkzeugen* zu erklären.

Im ersten Teil dieser Arbeit werden die relevanten Grundlagen aufbereitet. Dabei umfasst Kapitel 2 wesentliche domänenspezifische Referenzprozesse für die Entwicklung von Assistenz- und Automationssystemen sowie weitere Vorgehensmodelle, die den Stand der Technik und Forschung (Expertenwissen) widerspiegeln. Daraufhin werden in Kapitel 3 Beschreibungsmittel, Methoden und Werkzeuge eingeführt, die sich insbesondere mit dem Ziel der Verwebung von interdisziplinären Entwicklungsaktivitäten und der damit einhergehenden Formalisierung von Expertenwissen nutzen lassen.

Im zweiten Teil wird in Kapitel 4 präzisiert, was Interdisziplinarität im Kontext der Entwicklung von Assistenz- und Automationssystemen bedeutet und wie verschiedene Dimensionen in einem Entwicklungsprozesses berücksichtigt werden können. Darauf aufbauend werden die wesentlichen Teile der Domäne definiert sowie Anforderungen an ein Vorgehensmodell spezifiziert. Dieses wird in Kapitel 5 aufgegriffen, in welchem das verwobene Vorgehensmodell zusammen mit generischen Domänenmodellen beschrieben ist. Weiterhin wird die Prozessverifikation erklärt. Im nachfolgenden Kapitel 6 wird dargestellt, wie Methoden aus den verschiedenen Entwicklungssträngen miteinander verwoben werden können und wie dieses zur Unterstützung einzelner Entwickler innerhalb der verschiedenen Disziplinen genutzt werden kann. Dieses

[2] Quelle der Fahrzeugskizze in Abbildung 1-2:
`http://www.openclipart.org/detail/168/red-and-green-renault-twingo-by-molumen`

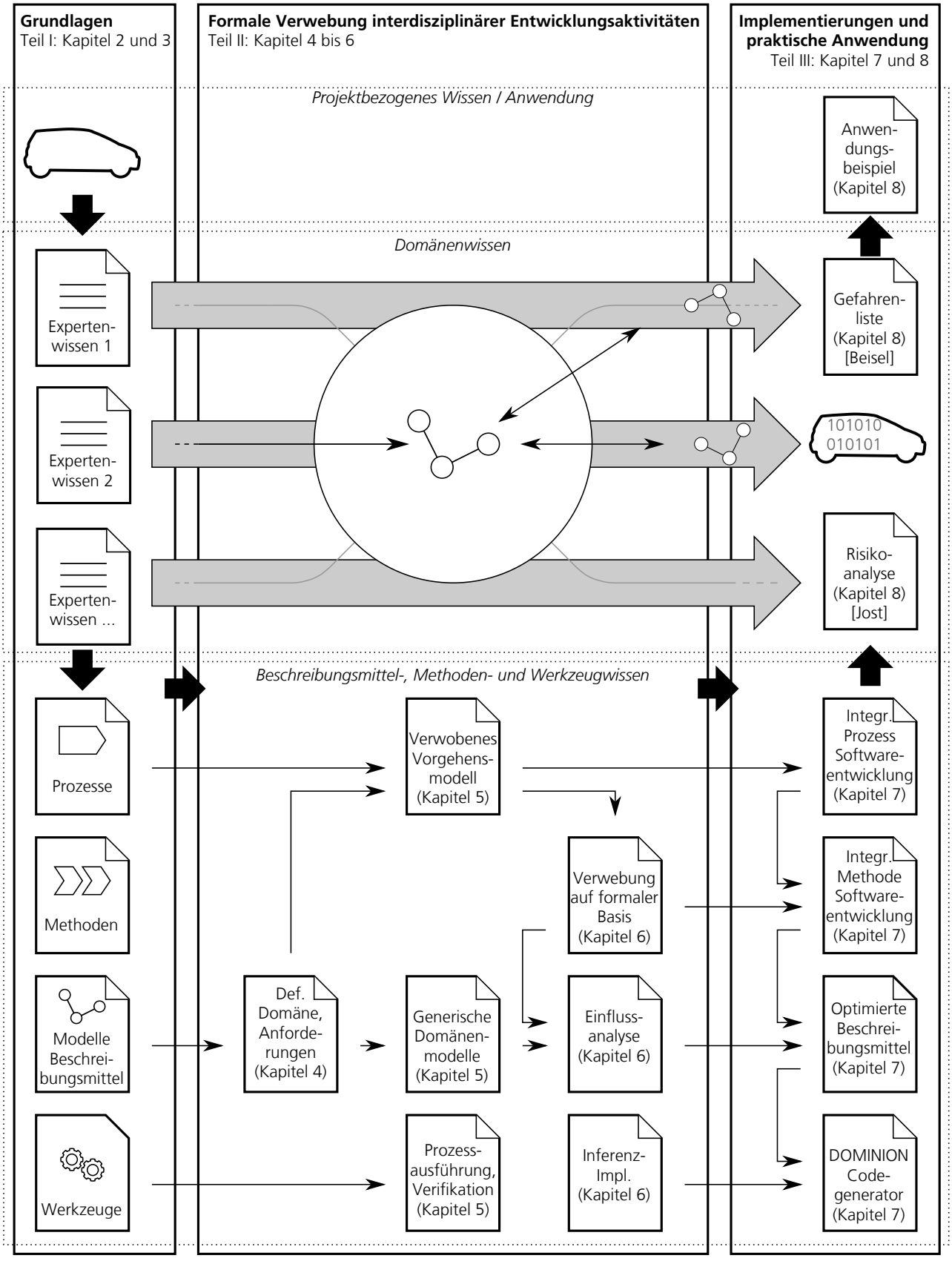

Abbildung 1-2: Struktur der Arbeit.

umfasst insbesondere eine Einflussanalyse bezüglich des Gesamtprozesses und wird durch prototypische Werkzeugimplementierungen dieser Arbeit gestützt.

Im dritten Teil steht die technische Umsetzung im Vordergrund. Dementsprechend wird in Kapitel 7 auf die Softwareimplementierung von Assistenz- und Automationssystemen eingegangen. Dazu wird beschrieben, wie diese mit den vorher definierten Prozessen und Methoden integriert ist. Zu diesem Zweck liefert diese Arbeit angepasste Softwarespezifikationssprachen sowie eine prototypische Umsetzung in einem Codegenerator. Weiterhin wird in Kapitel 8 eine Beispielanwendung dargestellt, welche die Erkenntnisse der vorangegangenen Kapitel einbezieht und verdeutlicht. Dazu wird in einer prototypischen Werkzeugkette die Weiterentwicklung eines ACC Systems zu einem Notbremssystem (EBS) beschrieben. Ergebnis ist ein ausführbarer Software-Demonstrator des Notbremssystems. Dabei werden in der technischen Umsetzung Modelle von Herrn Henning Jost und Herrn Daniel Beisel aus dem Projekt DeSCAS eingebunden.

Abschließend wird in einem vierten Teil die vorliegende Arbeit in Kapitel 9 zusammengefasst und bewertet. In diesem Zusammenhang werden auch Anknüpfungspunkte zur Fortführung der Thematik dieser Arbeit formuliert.

Teil I

Grundlagen

2 Vorgehensmodelle und Prozesse

2.1 Ziele des Kapitels

Das vorliegende Kapitel stellt Vorgehensmodelle und Prozesse für die Entwicklung von AAS vor. Dabei finden sich Referenzprozesse vor allem in relevanten Normen und Standards (Abschnitt 2.2). Von besonderem Interesse sind in diesem Zusammenhang vor allem Standards, die die funktionale Sicherheit von Assistenz- und Automationssystemen im Automobilbereich betreffen (Abschnitt 2.2.1). Dabei erfolgt eine kritische Reflexion dieser Referenzprozesse anhand verschiedener Kriterien, wie z.B. *Konsistenz*, *Korrektheit* oder *Interdisziplinarität*.

Im Rahmen der Darstellung wird unter anderem eine Strukturierung der Prozesse anhand von zwei der vier typischen Modellebenen für Metamodellierung nach [OMG06] vorgenommen, siehe auch Abbildung 2-1:

- *Modellebene* M_2: Die Abstraktionsebene M_2 (Metamodell) beschreibt, welche Konzepte für ein bestimmtes Modell zulässig sind und wie diese Konzepte zusammenhängen. Als Beispiel kann etwa das Konzept einer *Phase* genannt werden, welche durch weitere Phasen verfeinert werden kann.

- *Modellebene* M_1: Auf der Ebene M_1 (Modell) werden konkrete Instanzen der Konzepte aus M_2 beschrieben. So werden hier die Elemente eines konkreten Prozesses beschrieben. Dies kann z.B. eine Phase wie die *Systementwicklung* sein, welche durch die *Softwareentwicklung* verfeinert wird.

Daneben existieren die Modellebenen M_3 (Metametamodell) sowie M_0 (Modellinstanz). Auf der Ebene M_0 wird ein konkreter Prozess aus M_1 instanziiert. In diesem Kapitel handelt es sich dabei um die Durchführung konkreter Entwicklungsaktivitäten, wie etwa das Programmieren im Rahmen der *Softwareentwicklung*. Die Ebene M_3 umfasst abstrakte Konzepte, die eine Metamodellierung auf der Ebene M_2 ermöglichen. Dieses sind etwa *Klassen*, *Relationen* und *Attribute*. Ein standardisiertes Beschreibungsmittel hierfür ist die *Meta Object Facility* (MOF). Diese Strukturierung liefert wichtige Erkenntnisse über existierende Referenzmodelle für Entwicklungsprozesse, welches die Forschungsfragen weiter verfeinert. Darüber hinaus werden in Abschnitt 2.3 geeignete Vorgehensmodelle beschrieben, welche die Referenzmodelle ergänzen können. Dabei wird insbesondere herausgestellt, inwiefern sich diese für eine Anwendung im Rahmen der AAS Entwicklung anbieten.

2.2 Relevante Normen und Standards

Die 1998 eingeführte IEC 61508 [IEC03b] thematisiert die funktionale Sicherheit elektrischer, elektronischer und elektronisch programmierbarer Systeme. Sie stellt eine herstellerübergreifende Referenz für Vorgehensmodelle zur Entwicklung sicherheitskritischer Anwendungen zur Verfügung. Solche Referenzprozesse sind insofern besonders wichtig, da sie bei der Zulassung sicherheitskritischer Systeme als Bezug für den Stand der Forschung und Technik dienen.

Genau nach diesem müssen Produkte entwickelt werden, wenn ein Hersteller nicht Haftung für Schäden aus Fehlern seines Produktes übernehmen will. Dieses regelt das Produkthaftungsgesetz [Bun89]:

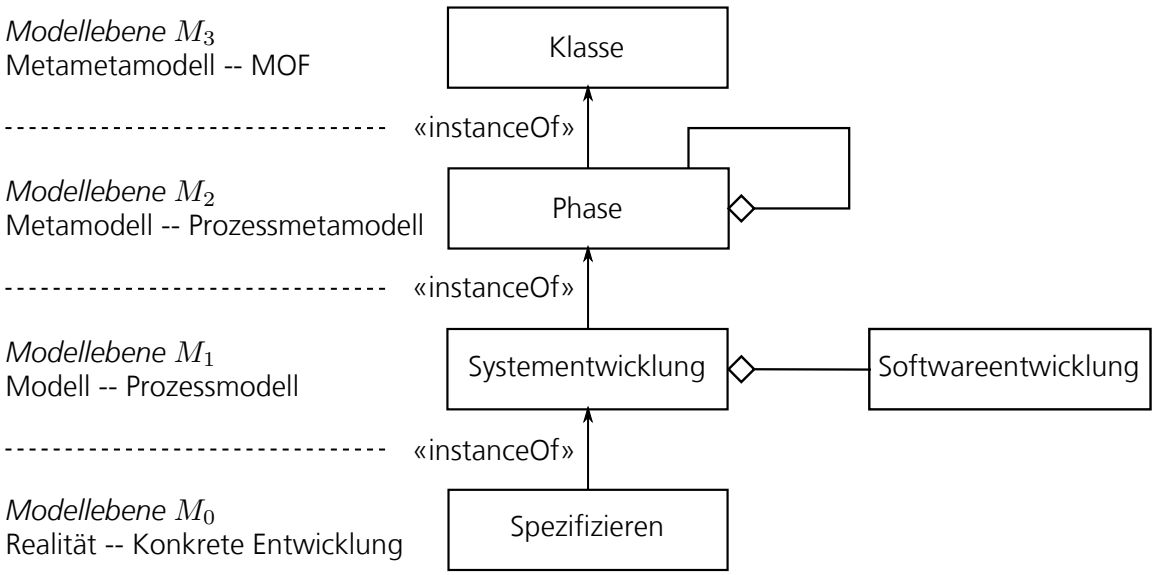

Abbildung 2-1: Auf der Basis eines abstrakten Metametamodells M_3 werden Metamodelle M_2 und Modelle M_1 spezifiziert, die reale Objekte in M_0 beschreiben.

> „Die Ersatzpflicht des Herstellers ist ausgeschlossen, wenn (. . .) der Fehler nach dem Stand der Wissenschaft und Technik in dem Zeitpunkt, in dem der Hersteller das Produkt in den Verkehr brachte, nicht erkannt werden konnte."

Die IEC 61508 ist eine Norm, die als domänenübergreifende Grundlage für die Entwicklung sicherheitskritischer Funktionen dient. Dazu werden sowohl ein Vorgehensmodell als auch konkrete Maßnahmen zur Unterstützung von Zertifizierungsprozessen von Komponenten und Systemen vorgeschlagen. Die zentrale Idee der Norm ist, dass durch die Markteinführung eines neuen Produktes ein definiertes gesellschaftlich akzeptiertes Risiko nicht überschritten werden darf. Dazu wird in einer Risikoanalyse festgestellt, ob und in welchem Maße dieses Risiko überschritten wird.

Darauf basierend werden dem System *Safety Integrity Level* (SIL) zugewiesen, an die wiederum risikomindernde Maßnahmen gekoppelt sind.

Für die Domäne der Bahnsysteme sind Derivate der IEC 61508 abgeleitet, wie etwa die EN 50126 [Eur97a] oder die EN 50128 [Eur97b]. Deutlich älter als die Zertifizierung im Automobilbau und der Bahnsysteme ist die Qualifizierung in der Luftfahrt, z.B. die DO-178B [RTC92].

2.2.1 Normen und Standards für Assistenz- und Automationssysteme

ISO DIS 26262

Die sich aktuell in der Normierung befindende ISO 26262 stellt einen sogenannten Applikationsstandard (oder Derivat) der IEC 61508 für den Automobilbau dar; im Sommer 2010 lag sie als *Draft International Standard* (DIS) [ISO09] vor. Dieser Standard verfeinert die Konzepte der Basisnorm und passt sie an die Domäne des Automobilbaus an. Wie das Prozessmodell der Basisnorm, ist auch das Prozessmodell der ISO DIS 26262 vom V-Modell inspiriert. Neben der Produktentwicklung, die als solches in einem V-Modell beschrieben ist, sind auch die Teilprozesse der Software- und Hardwareentwicklung ebenfalls in eigenen V-Modellen definiert. Des Weiteren legt die ISO DIS 26262 ein konkretes Vorgehen und Methoden für die Durchführung einer Risikoanalyse fest, welches von der Basisnorm abweicht. Die wesentlichen Einflussfaktoren sind dabei *probability of exposure* (E), *severity of potential harm* (S) und *controllability* (C). Dabei beschreibt der Faktor E, wie häufig von dem Produkt in der Entwicklung eine Gefahr ausgehen

kann. S drückt aus, wie groß der mögliche Schaden sein kann. Der Faktor C beschreibt, welcher Prozentsatz von Fahrern in der Lage ist, einen möglichen Schaden abzuwenden.

RESPONSE 3

Im Bezug auf die Normierung der ISO DIS 26262 wurde innerhalb des EU Projektes PreVENT vor allem der Aspekt der *controllability* (Kontrollierbarkeit) und dessen Nachweisbarkeit näher beleuchtet [Eur06b, Eur06a]. Weist man eine hohe Kontrollierbarkeit des Systems nach, müssen andere technische Maßnahmen zur Risikominderung nicht durchgeführt werden. Der RESPONSE 3 *Code of Practice* (CoP) gibt ebenfalls ein Prozessmodell vor, welches sich an einem Wasserfallmodell als Vorgehensmodell orientiert.

2.2.2 Strukturierung, Formalisierung und Kritik

Wie zuvor erläutert, können die in den Standards enthaltenen Vorgehensmodelle anhand von Modellebenen beschrieben werden. Dieses erleichtert schließlich ein Verständnis des zugrundeliegenden Prozesses. Nach dieser Modellierung findet eine kritische Reflexion statt.

Formalisierung: ISO DIS 26262

Betrachtet man die ISO DIS 26262 in Abbildung 2-2, so fallen auf der Modellebene M_1 vor allem die V-Modelle der System-, Software- und Hardwareentwicklung auf. Weiterhin wichtig ist der Teil 1 (Glossar), welcher nicht nur innerhalb des Standards für eine gemeinsame Begriffsbasis sorgt, sondern auch einen Einstieg in die funktionale Sicherheit bietet. Dazu lässt sich erkennen, dass die Organisation der funktionalen Sicherheit eine Querschnittsaktivität darstellt. Die funktionale Sicherheit hat insofern einen Einfluss auf die Funktionsentwicklung (System, Hardware, Software), als dass bestimmte entwicklungsbegleitende Maßnahmen umgesetzt werden müssen, wenn ein bestimmter *Automotive Safety Integrity Level* (ASIL) für das zu entwickelnde System bestimmt wurde. Dieses kann der Modellebene M_2 entnommen werden. Dabei fallen bei einem bestimmten ASIL sogenannte *Requirements & Recommendations* an, welche Anforderungen sind. Diese können sich entweder auf den Prozess oder auf das Produkt beziehen [1].

Der Norm kann dann weiterhin entnommen werden, mit Hilfe welcher Methoden und Maßnahmen welche Anforderungen erfüllt werden können. Somit bildet sie eine gute Grundlage, um als Leitfaden verwendet zu werden. Sie beantwortet die Frage, *wie* entwickelt werden muss, wenn gewisse Rahmenbedingungen vorliegen. Auf der Ebene der Modellebene M_3 kann man erkennen, dass dem Metamodell die grundlegenden abstrakten Konzepte der Nachvollziehbarkeit und des V-Modells zugrunde liegen.

Formalisierung: RESPONSE 3

Die Strukturierung von RESPONSE 3 CoP ist in Abbildung 2-3 dargestellt. Als wichtigster Unterschied ist auf Modellebene M_1 zu erkennen, dass hier nicht ein V-Modell als Basis vorliegt, sondern ein sequentielles Modell (Wasserfallmodell). Trotzdem haben die Phasen *Proof of Concept*, *Verification* und *Validation and Sign-Off* verifizierenden Charakter. Gleichzeitig sind genau zwei Iterationen skizziert. Das Metamodell in der Modellebene M_2 ist vergleichsweise einfach, ein hierarchisches sequentielles Vorgehensmodell. Im Anhang von RESPONSE 3 CoP sind ähnlich wie in der ISO Norm Methoden beschrieben, die etwa beim Nachweis der Kontrollierbarkeit zum Einsatz kommen. Stellt die entwicklerzentrierte Norm in den Mittelpunkt, *wie* bestimmte Anforderungen erfüllt werden können, so liegen im RESPONSE 3 CoP lose sortierte Methoden vor. Dieses macht die konkrete Auswahl schwierig, da keine explizite Verbindung zwischen Problemstellung (Anforderung) und Lösung (Maßnahme) existiert.

[1] Die Unterscheidung zwischen Prozess- und Produktanforderungen ist eine Interpretation der Norm nach [GJK+09] und dient der Kategorisierung der Anforderungen. Dasselbe gilt für die Kategorisierung der Methoden.

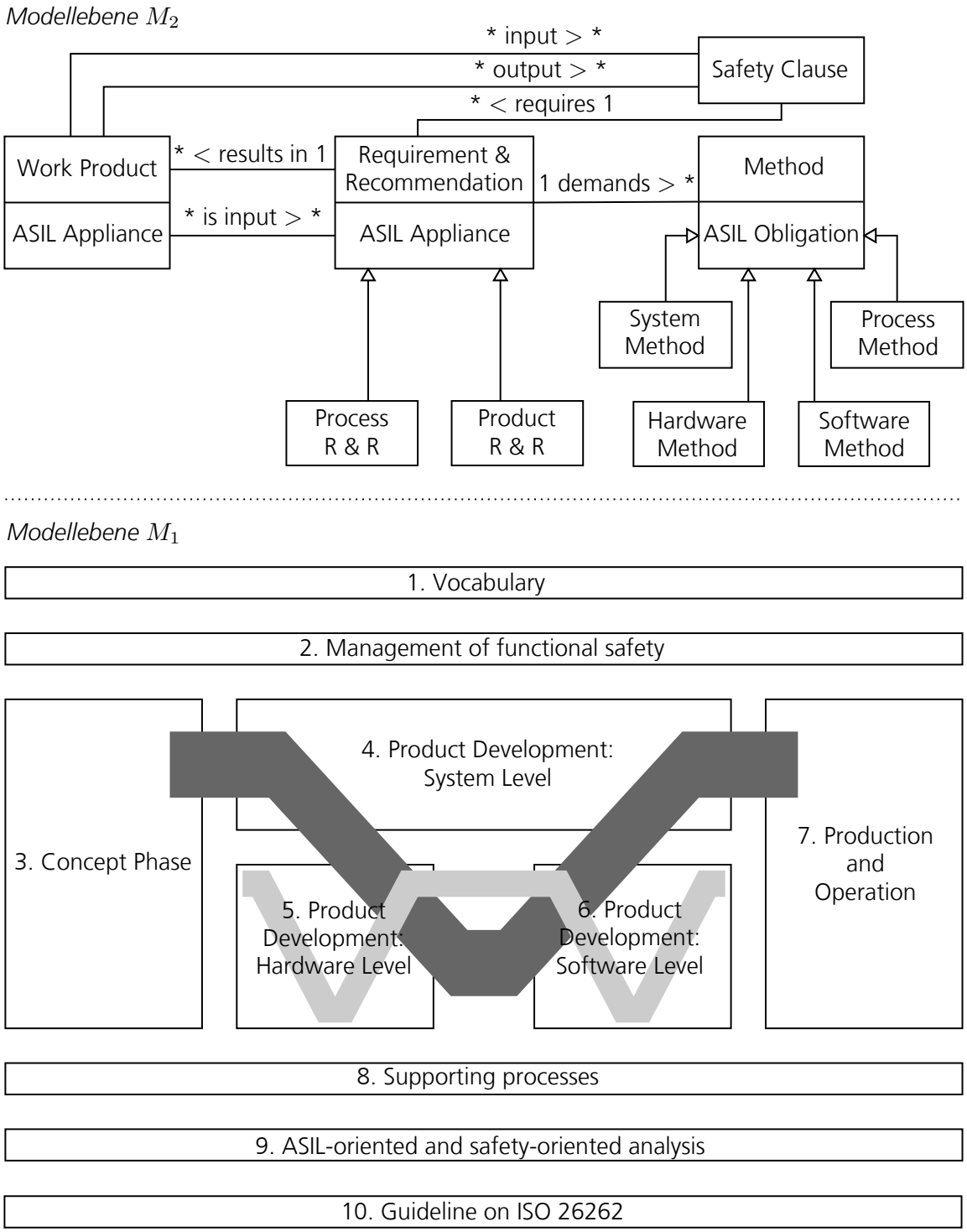

Abbildung 2-2: Das Modell der ISO DIS 26262 mit den drei verschachtelten V-Modellen (System, Software, Hardware), sowie einer Struktur, die einen Bezug zwischen Anforderungen sowie Methoden und Maßnahmen herstellt.

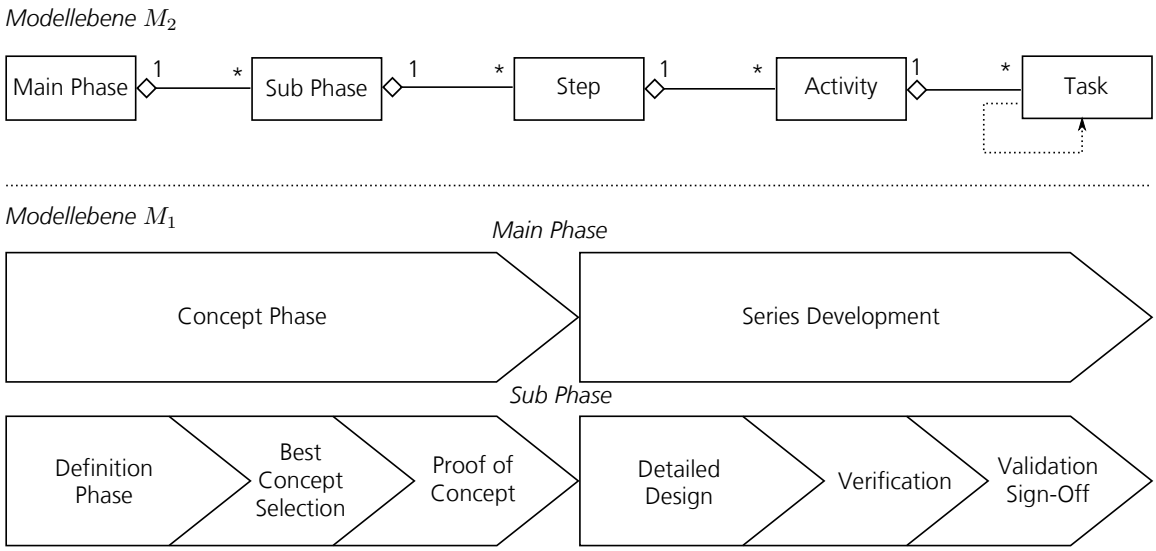

Abbildung 2-3: Das streng sequentielle Modell von RESPONSE 3 CoP, dabei sind auf M_1 die *Hauptphasen* und *Nebenphasen* dargestellt.

Kritische Reflexion: ISO DIS 26262 / RESPONSE 3

Betrachtet man die vorliegenden Referenzprozesse genauer, so werden Schwächen in den folgenden Bereichen offensichtlich (siehe auch [GJBK08, GJB+09]):

- *Interdisziplinarität*: Die genannten Standards gehen nur in sehr geringem Maße auf Aspekte der Arbeitsteilung und Interdisziplinarität ein. Dieses ist insofern als kritisch zu bewerten, als dass bei der Entwicklung von Assistenz- und Automationssystemen die komplexe Interaktion unterschiedlicher Disziplinen eine zentrale Rolle einnimmt. Lediglich in RESPONSE 3 CoP findet sich ein Hinweis, welche Entwicklungsschritte von welchen Disziplinen bearbeitet werden. Jedoch wird auch hier nicht auf die Interaktion eingegangen. Dieses wird auch dadurch deutlich, dass in den Prozessen zwar die Definition von Anforderungen eine Rolle spielt, nicht jedoch die Reflexion der Anwendungsdomäne an sich.

- *Konsistenz*: Die genannten Standards verfügen zwar über ein Glossar. Jedoch kommt es vor, dass einzelne Begriffe in verschiedenen Teilen desselben Dokuments verschieden verwendet werden [GJB+09]. Darüber hinaus ist kritisch zu bemerken, dass grundlegende Begriffe (wie etwa die *Kontrollierbarkeit* im RESPONSE 3 CoP) eine andere Bedeutung haben als in anderen relevanten Standards (etwa der ISO 17287 [ISO03a]). Zudem existiert keine Relation zu anderen in der Domäne relevanten Bedeutungen eines solchen Homonyms. Solche Relationen lassen sich etwa mit Hilfe von Ontologien abbilden. Dieses stellt den wesentlichen Mehrwert einer Ontologie im Gegensatz zu einer Taxonomie oder einem Glossar dar.

- *Korrektheit*: Die Erstellung eines solchen Standards in einem solchen informellen Format ist äußerst fehleranfällig. Die einfachste Form solcher Fehler sind „Tippfehler", wie sie sich etwa in RESPONSE 3 CoP finden. Beispielsweise ist *Aktivität 5.6.3.A*, die die Validierung und Genehmigung des neuen Systems umfasst, nicht in den Prozess eingebunden, da sie eine *Aktivität 5.6.2.D* als Voraussetzung hat. Diese existiert jedoch gar nicht im Prozessmodell. Solchen Fehlern kann dadurch begegnet werden, dass der Entwurf eines solchen Standards modellbasiert erfolgt. Dort würden die bisher vorliegenden Modellebenen $M_3 \ldots M_0$ formalisiert und dadurch prüfbar.

- *Abstrakte Syntax (Metamodell)*: Insbesondere die im Metamodell von RESPONSE 3 CoP verwendeten Begriffe, wie etwa *Phasen*, *Aktivitäten*, *Schritte*, *Aufgaben* sind sehr

missverständlich. Die Begriffe liegen sprachlich sehr nah beieinander und eine hierarchische Relation zwischen den Begriffen ist somit kaum nachvollziehbar. Der Differenzierung solcher Abstraktion lässt sich durch eine Formalisierung und Vereinfachung begegnen.

- *Konkrete Syntax (Präsentation)*: In der ISO DIS 26262 und insbesondere im Rahmen des RESPONSE 3 CoP werden Abläufe und Abhängigkeiten vor allem in strukturierter textbasierter Darstellung repräsentiert. Im RESPONSE 3 CoP etwa werden einzelne *Aufgaben* als Tabelle, Abhängigkeiten zu anderen Vorgängen durch eindeutige Schlüssel beschrieben. Dieses ist jedoch sehr unübersichtlich, die Korrektheit und Vollständigkeit von Abhängigkeiten lässt sich so nur sehr schwer prüfen. So finden sich im RESPONSE 3 CoP Fehler bei der Beschreibung von Abhängigkeiten, die bei Verwendung einer grafischen Notation schnell auffallen würden. Ein Beispiel ist hier die *Aktivität 5.5.2.A*, bei der die Mensch-Maschine Interaktion verifiziert wird. Die Durchführung dieser Phase ist jedoch nicht Voraussetzung für den Beginn späterer Phasen, so dass sie ausgelassen werden könnte. Weiteres Beispiel ist, dass die *Aktivität 5.2.3.A* (Definition eines Evaluationskriteriums) *nach Aktivität 5.2.3.B* (Auswahl des Konzeptes) erfolgen kann, was sicher nicht im Sinne der Autoren ist. Schon bei der hier vorliegenden Beschreibung wird schnell klar, dass eine solche Repräsentation von logischen Verknüpfungen sehr fehleranfällig ist.
- *Offenheit*: Die ISO DIS 26262 basiert auf einem V-Modell, RESPONSE 3 CoP auf einem sequentiellen Modell (Wasserfallmodell). Gerade weil beide Prozesse sehr verschiedene Metamodelle aufweisen, ist es ohne formalisierte Modellierung schwierig, beide Prozesse miteinander konsistent zu verweben. Eine Verwebung kann vor allem nur dann erfolgreich sein, wenn solch eine formalisierte Modellierung offen gestaltet ist, also ein transparenter Zugang zu den Modellen der Ebenen M_2 und M_1 vorliegt.
- *Erweiterbarkeit*: Da es sich um Referenzprozesse handelt, ist davon auszugehen, dass es sich um idealisierte Prozesse handelt, und so wie beschrieben nie unangepasst instanziiert werden. Vielmehr ist es wichtig, unternehmenseigene Prozesse von solchen Referenzprozessen abzuleiten oder alternativ eigene Prozesse auf einen Referenzprozess zu beziehen. Dabei kann ein offenes, formalisiertes Modell helfen. Gleichzeitig wäre dieses ein erster Schritt zum Herstellen von *Traceability* (Nachvollziehbarkeit) zwischen Referenzprozess und Unternehmensprozess, was die Grundlage für eine effiziente Nachweisführung bildet.

2.3 Grundlegende Vorgehensmodelle und Entwicklungsphasen

In diesem Abschnitt wird ein Überblick über die wichtigsten und meist verwendeten Vorgehensmodelle bei der System- und Softwareentwicklung gegeben. Diese werden in Kategorien zusammengefasst dargestellt (angelehnt an [BvK08]). So wird ein Problembezug hergestellt, über den die Auswahl einer passenden Vorgehensweise unterstützt werden soll. Diese Kategorisierung ist in Abbildung 2-4 dargestellt.

2.3.1 Kategorien elementarer Vorgehensmodelle

Streng sequentielle Vorgehensmodelle

Bei einem solch *linearen* Vorgehensmodell laufen Phasen streng sequentiell ab, die Wiederholung einer Phase oder der Rückschritt auf eine frühere Phase lässt dieses Vorgehensmodell nicht zu. Das bekannteste Modell ist hier das *Wasserfallmodell* [Roy70] ohne Wiederholungen von Phasen. Solch streng sequentielles Vorgehen bildet reale Prozesse nur schlecht ab, wird jedoch bei der Erstellung von Projektplänen nach wie vor am häufigsten verwendet. Geeignet ist dieses Vorgehen nur für gut a priori planbare, kleine Projekte. Vorteil dieses Vorgehens liegt in seinem

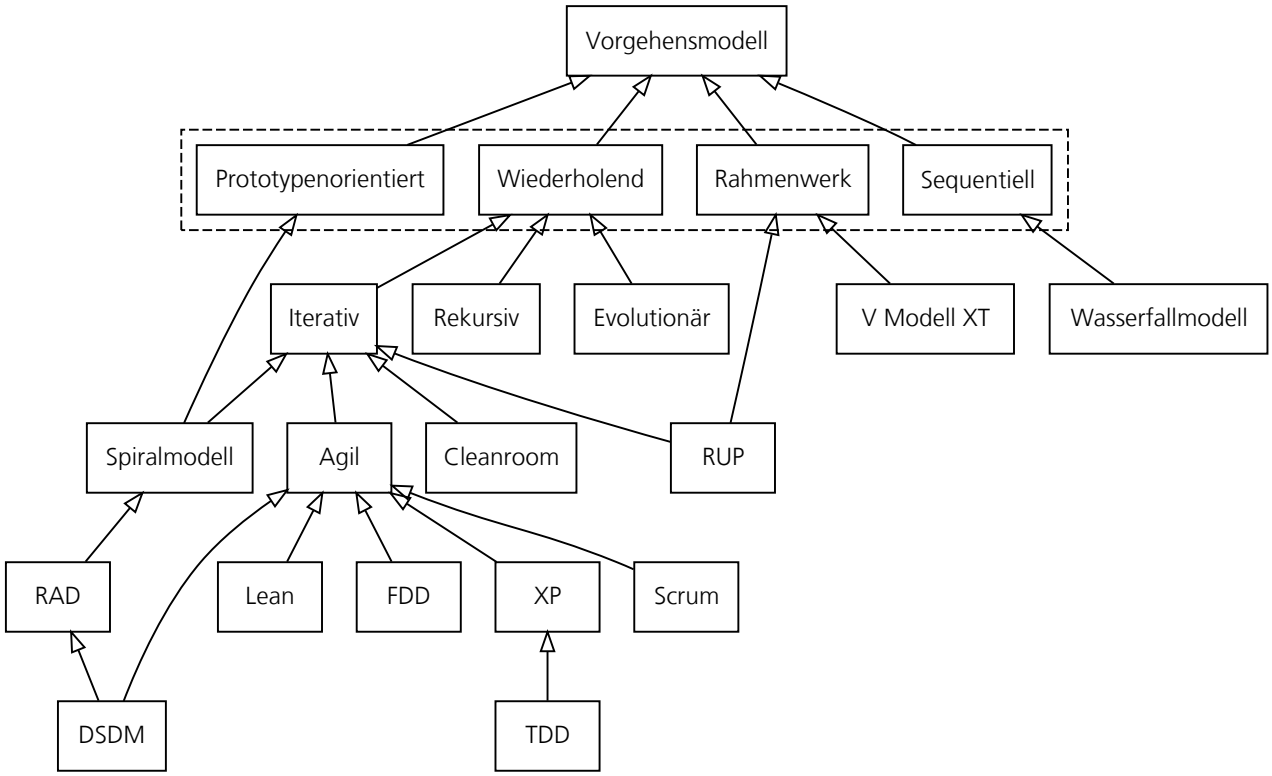

Abbildung 2-4: Grundlegende Vorgehensmodelle, die Kategorien sind an [BvK08] angelehnt.

einfachen Verständnis. Im Kontext der Entwicklung von AAS kann dieses eigentlich nur für kleine Teilprojekte innerhalb der Entwicklung gelten.

Prototypenorientierte Vorgehensmodelle

Bei den prototypenorientierten Vorgehensmodellen werden bewusst anhand eines Prototyps Anforderungen eines (Teil-)Systems diskutiert, wobei auch ein Kunde einbezogen werden kann. Im Gegensatz zu den wiederholenden Vorgehensmodellen sind jedoch die Anzahl an Konfigurationen und Versionen relativ gering. Diese Klasse von Vorgehensmodellen eignet sich somit gut für Serienentwicklungsprojekte neuartiger AAS. Ein bekanntes prototypenorientiertes Modell ist das *Spiralmodell* [Boe86]. Ein explizit prototypenorientiertes Vorgehen ist *Rapid Application Development* (RAD) [Mar91].

Wiederholende Vorgehensmodelle

Ein wichtiger Unterschied solch *iterativer*, *rekursiver* oder *evolutionärer* Modelle zu sequentiellen Vorgehensmodellen ist, dass Phasen wiederholt werden können. Obwohl das Ziel des Projekts formuliert ist, geht man davon aus, dass zu Projektstart noch nicht alle Anforderungen bekannt sind. Der Vorteil liegt darin, dass während des Projektes gelernt werden kann. Somit wirken sich kritische Fehler in der Projektplanung weitaus weniger stark aus. Diese Klasse von Vorgehensmodellen eignet sich daher gut für große Serienentwicklungsprojekte oder Forschungsprojekte von AAS. Der bekannteste Vertreter ist auch hier das *Spiralmodell* [Boe86]. Auch der *Rational Unified Process* (RUP) [BJR99] weist wiederholendes Vorgehen auf, wie auch das *Cleanroom Software Engineering* [MDL87], welches besonders stark auf den Einsatz formaler Methoden in der Entwicklung setzt.

Agile Vorgehensmodelle (welche den Entwickler über Prozesse und Methoden setzen) sind etwa *Scrum* [Sch04], *Extreme Programming* (XP) [Bec99], *Lean Software Development* [PP03], *Feature Driven Development* (FDD) [CLL99], *Test Driven Development* (TDD) [Bec02] und auch *Dynamic Systems Development Method* (DSDM)[2]. Sie sind jedoch nicht besonders als

[2] `http://www.dsdm.org/`

globales Vorgehensmodell für Serienentwicklungsprojekte von AAS geeignet, da vor allem bei sicherheitskritischen Systemen komplexe Prozesse eingehalten werden müssen, um die Haftung beschränken zu können. Jedoch können Methoden wie XP eingesetzt werden, wenn es sich um implementierungsnahe Phasen im Feinentwurf von Software handelt.

Rekursives Vorgehen ist dem iterativen sehr ähnlich, dabei steht die vertiefende, genaue Wiederholung einer oder mehrerer Entwicklungsaktivitäten im Vordergrund.

Verglichen mit dem iterativen Vorgehen bietet das evolutionäre Vorgehen noch mehr Flexibilität. Neben der Iteration des Entwurfs werden hier sogar die Projektziele kritisch beleuchtet bzw. sind unter Umständen zu Projektstart noch gar nicht bekannt. Da Ziele üblicherweise nicht klar benannt werden können, ist hier auch das Risiko einer Zielverfehlung am geringsten. Jedoch geht ein solches Vorgehen insbesondere zu Lasten der Effizienz. Evolutionäres Vorgehen im Kontext der Entwicklung von AAS bietet sich somit insbesondere im Bereich der Grundlagenforschung an, wo der Erkenntnisgewinn im Vordergrund steht.

Vorgehensrahmenwerke / Tailoring

Bei einem Vorgehensrahmenwerk steht nicht so sehr die Art des Vorgehens oder die Abarbeitungsfolge von Phasen im Vordergrund, sondern eher die flexible Komposition von Komponenten eines Vorgehensmodells. Dabei beschreibt etwa das *V-Modell XT* [Die09] keinen zeitlichen Ablauf zwischen verschiedenen Phasen, sondern nur logische Abhängigkeiten von (Teil-)Produkten. Das besondere liegt dabei in der Idee der Definition eines Vorgehensrahmenwerkes, welches auf Bedarf zu einem konkreten Vorgehensmodell zugeschnitten werden kann. Dieser Schritt wird als *Tailoring* bezeichnet. Daraus leitet sich auch die Abkürzung für *eXtreme Tailoring* (XT) ab. Eine weitere Eigenheit des V-Modells liegt in der Tatsache, dass jedem kreativen Schritt ein entsprechender verifizierender Schritt gegenübergestellt wird. Die wesentlichen Ergänzungen des V-Modell XT gegenüber dem V-Modell 97 [Die97] liegen im Tailoring und der Modellierung von Auftragsanbahnungsphasen. Das V-Modell ist das am weitesten verbreitete Vorgehensmodell in der Serienentwicklung im Automobilbereich [SZ03]. Ein weiterer Vertreter der Vorgehensrahmenwerke ist der *Rational Unified Process (RUP)* [BJR99], welcher insbesondere das Vorgehen mit bestimmten Werkzeugen und Diensten verknüpft.

2.3.2 Elementare Entwicklungsphasen

An dieser Stelle werden wesentliche basale Entwicklungsphasen beschrieben, wie sie als Dogma in der Literatur nach [Bjø08, Bjø10] zu finden sind:

> „Vor dem Software-Entwurf müssen wir dessen Anforderungen kennen. Vor der Formulierung von Anforderungen müssen wir die Domäne verstehen."

Dieses Dogma wird insbesondere deshalb hier beschrieben, da es die Modellierung einer Domäne als Phase vor das Requirements Engineering stellt. Dieses kann als Basis für eine interdisziplinäre Zusammenarbeit dienen. Des Weiteren beschreibt [Bjø08] eine Formalisierung, die sukzessive natürlichsprachliche Modelle in mathematische Modelle überführt.

Domain Engineering (1)

Zum *Domain Engineering* nach [Bjø10] gehört die Akquisition von Domänenwissen, der dazugehörigen Analyse und Modellierung sowie einer Verifikation und Validierung der Modellierung des Domänenwissens. Im Kontext interdisziplinärer Systementwicklung kann diese Phase dazu dienen, eine *fremde* Domäne zu verstehen. Gleichzeitig können die entstehenden Modelle genutzt werden, Anforderungen zu annotieren und bezüglich der verwendeten Begriffe verständlich zu machen.

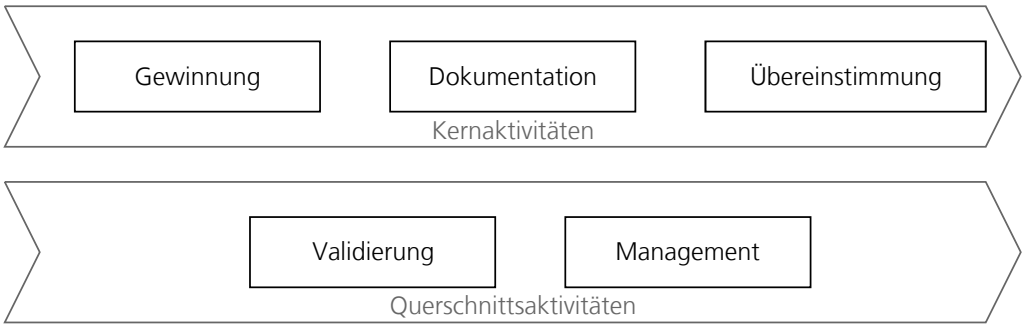

Abbildung 2-5: Wesentliche Phasen im Requirements Engineering nach [Poh08].

Das Domain Engineering bedeutet als zusätzliche Entwicklungsaktivität erhöhten Aufwand. Dieser kann sich lohnen, wenn durch das Domänenmodell die nachfolgenden Phasen effizienter durchlaufen werden können. Weiterhin kann das Domain Engineering dazu dienen, den Grad an Wiederverwendung von Entwurfsartefakten zu erhöhen und somit über einzelne Produktentwicklungen hinweg Entwicklungskosten einzusparen.

Daher wird das Domain Engineering auch häufig in einem nicht explizit interdisziplinären Umfeld eingesetzt. So kann es genutzt werden, wenn verschiedene ähnliche Produkte [MJET09] geplant sind, oder ein Produkt in mehreren Varianten entwickelt wird. In diesem Zusammenhang ist für das Domain Engineering auch die Bezeichnung *Product Line Engineering* gängig [FK05].

Requirements Engineering (2)

Nach [Bjø10] werden beim *Requirements Engineering* Stakeholder identifiziert, Anforderungen entwickelt, modelliert und analysiert. Anschließend werden Anforderungen validiert sowie Machbarkeits- und Erfüllbarkeitsstudien durchgeführt. Dazu ähnlich, jedoch stärker strukturiert werden bei [Poh08] drei Kernaktivitäten benannt (*Dokumentation*, *Gewinnung* und *Übereinstimmung*) sowie zwei Querschnittsaktivitäten (*Validation* und *Management*), wie in Abbildung 2-5 dargestellt. Ferner ist diese Beschreibung ähnlich dem standardisierten *Stakeholder Requirements Definiton Process* aus der ISO 15288 [ISO08].

Eine wesentliche Änderung im Requirements Engineering durch das Domain Engineering ist, dass nach [Bjø10] Kategorien von Anforderungen anfallen. Durch diese Kategorisierung (und somit Strukturierung) erhöht sich vor allem der Grad an Wiederverwendung von Anforderungsartefakten. Denn Eigenschaften einer Domäne, und daraus resultierende Anforderungen (beispielsweise Verkehrsregeln), verfügen über eine hohe Stabilität verglichen mit direkten Kundenanforderungen.

Das Requirements Engineering spielt bei der Verwebung interdisziplinärer Entwicklungen eine zentrale Rolle. Dabei werden Anforderungen aus allen relevanten Domänen an das Produkt festgelegt. Somit erfolgt auch ein großer Teil der Interaktion und Abstimmung zwischen den beteiligten Disziplinen über die Ebene der Anforderungen. So kann etwa über Sicherheitsanforderungen an die technische Funktionsentwicklung Einfluss auf die Sicherheitseigenschaften des Produkts genommen werden. Daher müssen Anforderungen von allen beteiligten Entwicklern verstanden werden, damit sowohl das *was* als auch das *warum* klar ist.

Entwurf von Architekturen und Funktionen (3)

Elementare Schritte sind die Definition einer *Architektur*, darauf folgt jeweils ein entsprechender *Funktionsentwurf*. Dieses betrifft System, Hardware und Software. Der Entwurf greift dabei die Anforderungen auf, die auf Basis von Domänenwissen formuliert wurden. Dazu gehört eine verfeinerte Spezifikation von funktionalen wie extrafunktionalen Eigenschaften des vorliegenden Entwicklungsgegenstands. Somit stellt das Ergebnis eines Entwurfs die Spezifikation für die

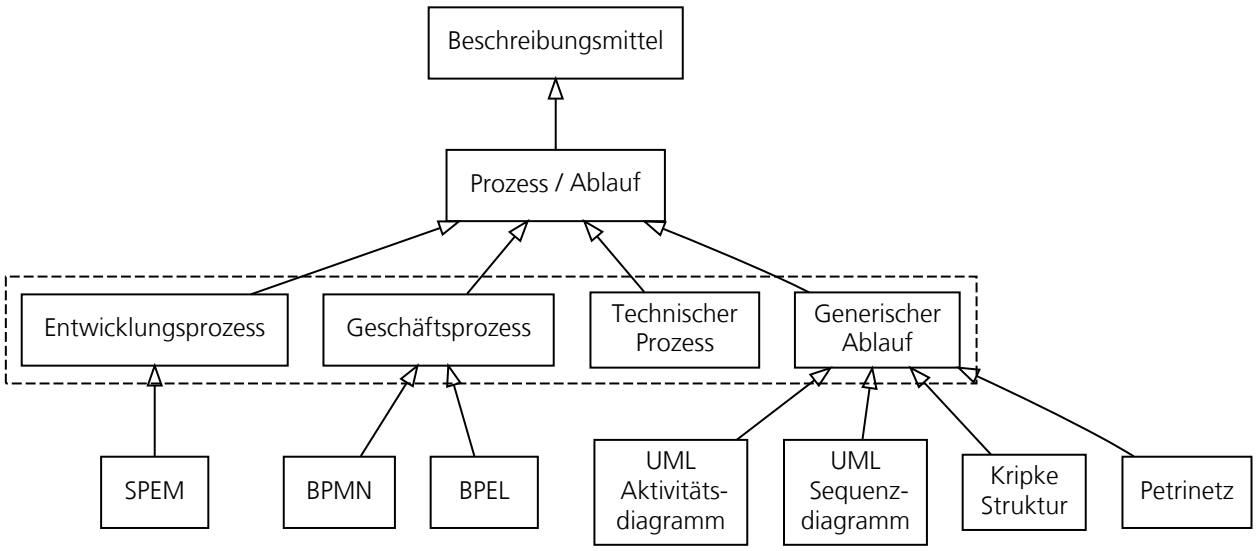

Abbildung 2-6: Eine Auswahl an Beschreibungsmitteln, wie sie im Kontext der Automatisierungstechnik vorliegen.

jeweils folgenden Entwurfsaktivitäten und kann als Rekursion von Domain Engineering (1) und Requirements Engineering (2) betrachtet werden.

2.3.3 Beschreibungsmittel für Prozesse und zeitliche Abläufe

In der Domäne der Automatisierung liegen eine Reihe verschiedener Beschreibungsmittel vor. Die meisten verfügen über eine semiformale Basis. Klassisch aus der Domäne der Automatisierungstechnik sind *Funktionsbausteine* nach IEC 61131 [IEC03a], *Funktionsblöcke* nach IEC 61499 [IEC05] sowie die Beschreibung von *Prozessleittechnik* nach DIN 19227 [Deu91] und *Regelungs- und Steuerungstechnik* nach DIN 19226 [Deu95].

Ähnlich generisch kommen aus dem Umfeld der Softwareentwicklung Diagramme der *Unified Modeling Language* (UML), siehe Abbildung 2-6. Dabei handelt es sich insbesondere um das *Aktivitätsdiagramm* [OMG09c] und das *Sequenzdiagramm* [OMG09c]. Generisch einsetzbar mit formal beschriebener Syntax und Semantik sind *Petrinetze* [Pet62, WP08] sowie auch *Kripke Strukturen* (siehe Anhang B.1), die als Systemmodelle im Rahmen formaler Verifikation sehr geläufig sind. Diese können gegen Spezifikationen in Form temporaler Logik geprüft werden, wie beispielsweise *Linear Temporal Logic* (LTL) oder auch *Computation Tree Logic* (CTL).

Die genannten Beschreibungsmittel sind universell verwendbar, indem sie Abläufe abbilden, jedoch nicht konkrete Typen von Prozessen (beispielsweise einen Entwicklungsprozess) modellieren. Solche mit einer formalen Basis sind jedoch geeignet, bestimmte Eigenschaften eines Vorgehensmodells zu prüfen (beispielsweise die Erreichbarkeit bestimmter Phasen). Jedoch eignen sie sich nur bedingt für die Prozessmodellierung, da sie nicht die Konzepte eines Vorgehensmetamodells abbilden (Phasen, Meilensteine).

An dieser Stelle kommen Beschreibungsmittel zum Einsatz, die sich explizit mit Geschäfts- oder Entwicklungsprozessen auseinandersetzen. Dazu zählen etwa die OMG Standards *Software & Systems Process Engineering Meta-Model* (SPEM) [OMG08c], *Business Process Model and Notation* (BPMN) [OMG09a] oder auch der OASIS Standard *Business Process Execution Language* (BPEL) [JE07].

2.4 Zusammenfassung des Kapitels

In Abschnitt 2.2 wurden wesentliche Referenzprozesse im Bereich von (sicherheitskritischen) Assistenz- und Automationssystemen im Automobil beschrieben. Diese wurden anhand der Modellebenen M_1 und M_2 aufbereitet und die wesentlichen Verbesserungsmöglichkeiten in den Bereichen Interdisziplinarität, Konsistenz, Korrektheit, Abstrakte Syntax, Konkrete Syntax Offenheit und Erweiterbarkeit identifiziert. Insbesondere die Einführung der ISO DIS 26262 bietet die Chance, Entwurfsentscheidungen abzusichern.

Die Anwendung des Standards ermöglicht eine gute Orientierung für die Entwicklung sicherheitskritischer Systeme. Potential zur Verbesserung liegt hier vor allem in der Formalisierung des Standards. Denn er besteht aus mehreren 100 Seiten Papier, die aufwändige Einarbeitung erforderlich machen und eine Prüfung bezüglich Konsistenz, Korrektheit und Vollständigkeit erschweren. Hinzu kommt vor allem, dass durch den umfassenden Inhalt der Norm eine Anwendung nur mit sehr viel Erfahrung und Lernaufwand möglich ist. An dieser Stelle lohnt sich eine Verknüpfung mit konkreten Problemen der Anwendungsdomäne, um insbesondere die Komplexität und den irreführenden Interpretationsspielraum für Funktionsentwickler zu reduzieren.

Weiterhin wurden in Abschnitt 2.3 wesentliche Kategorien von Vorgehensmodellen und deren Eignung für die AAS Domäne beschrieben. Für die Entwicklung eines komplexen, sicherheitskritischen Systems eignen sich Vorgehensrahmenwerke (wie etwa das V-Modell XT), da nur durch das Tailoring eines Referenzprozesses wie der ISO 26262 die Prozesskonformität für einen Sicherheitsnachweis sinnvoll nachgewiesen werden kann. Ergänzt werden sollte ein *globales* Rahmenwerk durch den *lokalen* Einsatz agiler Vorgehensweisen. Diese ermöglichen hohe Flexibilität und Qualität bei der konkreten Umsetzung.

Darüber hinaus wurden die wesentlichen Entwicklungsphasen des *Domain Engineering*, *Requirements Engineering* und *Entwurf* eingeführt. Diese sind für die folgenden Kapitel wichtig. Dabei liegt die Bedeutung des Domain Engineering vor allem in der Beschreibung der Anwendungsdomäne, beim Requirements Engineering findet die konkrete Interaktion zwischen den Disziplinen über Anforderungen statt.

Wichtig für die Formalisierung und Modellierung eines Vorgehensmodell sind die in Abschnitt 2.3.3 vorgestellten Beschreibungsmittel für Prozesse. Dabei ist die Unterscheidung zwischen dem konzeptionellen Modell (abstrakte Syntax) des konkreten Vorgehensmodell und der Modellierung temporaler Semantik wichtig. Insbesondere Automaten (z.B. Kripke Strukturen) im Zusammenhang mit temporaler Logik sind nach dem Stand der Technik für eine formale Modellierung temporaler Abläufe geeignet. Diese Modelle und Logiken lassen sich vor allem auch deshalb gut einsetzen, da ausgereifte Analysewerkzeuge (z.B. Model Checker) vorliegen.

3 Domänenwissen und Anforderungsmodellierung

3.1 Ziele des Kapitels

Nach der Beschreibung von Vorgehensmodellen und Prozessen stehen in diesem Kapitel die wissensbasierte Modellierung von Domänenwissen sowie die Modellierung von Anforderungen im Mittelpunkt.

In Abschnitt 3.2 werden zunächst Methoden beschrieben, die den Einsatz bestimmter Beschreibungsmittel und Werkzeuge motivieren. Dabei wird insbesondere ausgearbeitet, welche Methoden sich für die interdisziplinäre Systementwicklung eignen. Der Schwerpunkt liegt auf Methoden des *Domain Engineering* und vor allem auf dem *Requirements Engineering*. Ein wichtiges Kriterium ist dabei die *Formalisierbarkeit*.

Die Formalisierung von Methoden erfolgt mit Hilfe angemessener Beschreibungsmittel, die in Abschnitt 3.3 erläutert werden. Hier finden sich Metamodelle zu den vorher beschriebenen Methoden. Darüber hinaus werden vor allem Beschreibungsmittel eingeführt, die sich als gemeinsame technische Grundlage für eine offene Formalisierung eignen. Hierbei ist die *Web Ontology Language* (OWL) hervorzuheben, die sich zur Modellierung von Domänenwissen eignet.

Abschließend beschreibt Abschnitt 3.4 Werkzeuge, die eine Modellierung der genannten Beschreibungsmittel ermöglichen und somit die erläuterten Methoden sowie die Formalisierung unterstützen.

3.2 Methoden

Im Folgenden werden Methoden beschrieben, die zur Modellierung von Domänenwissen bzw. Anforderungen verwendet werden können.

3.2.1 Domänenmodellierung

Unter dem Gesichtspunkt der Wiederverwendung existieren verschiedene Methoden für das Domain Engineering, etwa *Domain Analysis and Reuse Environment* (DARE) [FPDF98], *Family-oriented Abstraction, Specification, and Translation* (FAST) [WL99], *Feature-Oriented Reuse Method* (FORM) [KLD02], *Komponentenbasierte Anwendungsentwicklung* (KobrA) [ABB+01], *Product Line UML-based Software engineering* (PLUS) [Gom04] oder *Koala* [vOvdLKM00]. Einen genaueren Überblick gibt es bei [FK05]. Dabei umfasst das *FAST* Modell zusätzlich zum Domain und Application Engineering die sogenannte *Domain Qualification*, die darüber entscheidet, ob sich die Entwicklung einer neuen Domäne (Produktlinie) betriebswirtschaftlich lohnt.

Mit dem Fokus auf der Verwebung interdisziplinärer Entwurfsaktivitäten kommt dem Domain Engineering jedoch insbesondere die Rolle des Verstehens einer fremden Domäne zu. Dieses umfasst im einfachsten Fall die Definition von Domänenwissen in Form von Begriffen. So wird etwa bei [Sch08] ein Vorgehen beschrieben, was zur Begriffsdefinition verwendet werden kann. Dabei wird im Wesentlichen ein Begriff sowie dessen Relationen formalisiert und anschließend verifiziert.

Auch im Bereich der Erstellung von Ontologien existieren eine Reihe von Methoden, die bei [CC05] verglichen werden.

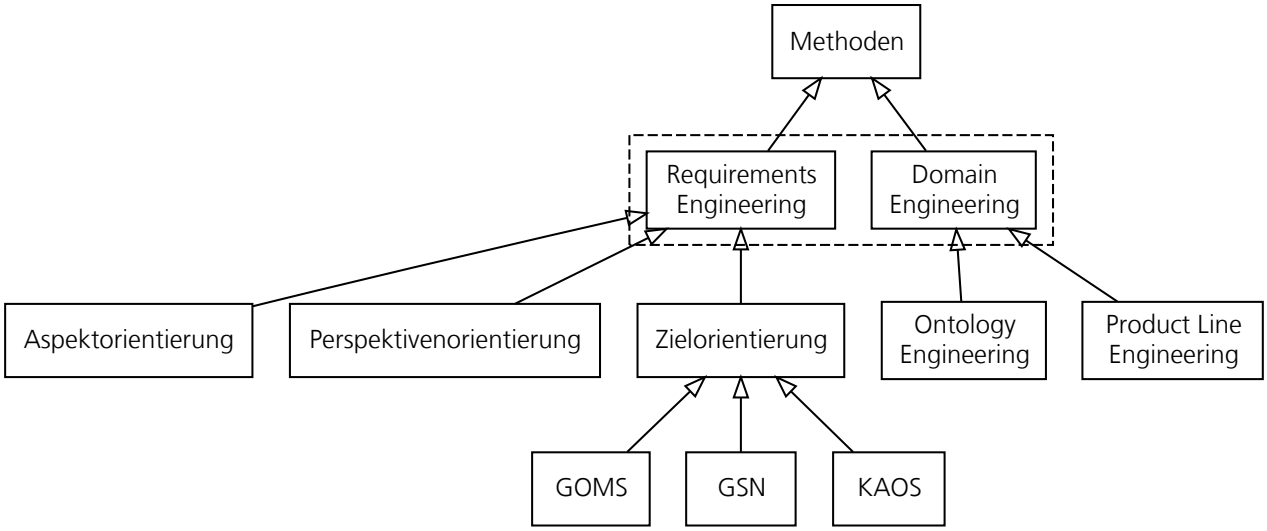

Abbildung 3-1: Eine Auswahl an Methoden des Domain und Requirements Engineering, welche sich im Kontext der Interdisziplinarität anbieten.

3.2.2 Anforderungsmodellierung

Im Weiteren werden Methoden beschrieben, die sich insbesondere vor dem Hintergrund der Interdisziplinarität und Formalisierbarkeit für einen Einsatz eignen. Darunter fallen vor allem Methoden, die sich an *Perspektiven*, *Aspekten* oder *Zielen* orientieren. Diese sind in Abbildung 3-1 visualisiert.

Orientierung an Perspektiven (Viewpoints)

Perspektiven spielen beim Requirements Engineering vor allem dann eine Rolle, wenn es um die Aufdeckung und Lösung von Konflikten in Anforderungsdokumenten geht. Dabei argumentiert [Eas91], dass verschiedene Perspektiven auf das zu entwickelnde System erst einmal getrennt voneinander modelliert werden müssen. Erst danach werden sie in eine gemeinsame konsistente Spezifikation überführt. Geschieht dieses nicht, geht zum einen die Nachvollziehbarkeit verloren und weiterhin kann es zu Inkonsistenzen in der Spezifikation führen.

Bei [FF89] wird beschrieben, wie agentenbasiert zwei formal modellierte Perspektiven kooperativ zu einer gemeinsamen und konsistenten Spezifikation kommen. Bei [KS96] werden *direkte* und *indirekte* Perspektiven unterschieden, wobei sich die direkte Form auf einen unmittelbaren Nutzer bzw. Teil des zu entwickelnden Systems bezieht. Indirekte Perspektiven interagieren nicht unmittelbar mit dem zu entwickelnden System.

Über die Orientierung an Perspektiven und deren Modellierung wird insbesondere die Nachvollziehbarkeit zwischen Anforderungen und einem jeweiligen *Stakeholder* sichergestellt. Im Laufe einer funktionalen Dekomposition werden so Entwurfsentscheidungen besser verständlich und außerdem spezifisch dokumentiert.

Orientierung an Aspekten

Sowohl die Orientierung an Zielen und Perspektiven folgt üblicherweise dem Paradigma des *divide and conquer* oder *seperation of concerns*, um die Komplexität des Problems beherrschbar zu machen. Die der Trennung zugrunde liegende Modellvorstellung ist jedoch immer idealisiert, da sich weder Perspektiven vollständig isolieren lassen, noch sich eine reale hierarchische Zielstruktur finden lässt. Somit verbleiben sogenannte *cross cutting concerns*, Querschnittsaspekte. Ziel des aspektorientierten Requirements Engineering ist genau die Identifikation solcher Abhängigkeiten, die etwa zwischen verschiedenen Zielen und Perspektiven existieren. Dabei können Anforderungen aufgedeckt werden, die bei einer isolierten Sichtweise nach Zielen oder Perspektiven nicht entdeckt werden. Der Ansatz nach [YdPLM04] identifiziert

möglichst früh Aspekte. Aspektorientierung kommt aus dem Softwareentwurf, kann aber auch schon im Requirements Engineering eingesetzt werden [Kai08].

Zielorientierte Erhebung von Anforderungen
Der zielorientierten Erhebung von Anforderungen liegt die explizite Erhebung und Formulierung von Bedürfnissen der Stakeholder zugrunde (diesbezüglich ähnlich der Orientierung an Perspektiven). Bezogen auf den Nutzen, die die Formulierung von Zielen bei der Erhebung von Anforderungen hat, ist der Aufwand vergleichsweise gering [Poh08]. [Hei05] liefert einen Überblick und Vergleich von zielorientierten Methoden zur Erhebung von Anforderungen. Nach dieser Bewertung ist die KAOS Methode die insgesamt beste Methode zielorientierter Erhebung und Spezifikation von Anforderungen. Die Eignung der einzelnen Methoden hängt jedoch stark von der Phase der Anwendung ab (z.B. Gewinnung, Dokumentation).

In Abhängigkeit von der konkreten Methode kann die Definition eines Ziels variieren. Die Definition nach [vL01] lautet:

> „Ein Ziel ist eine Vorgabe, die das betrachtete System erfüllen sollte. Die Formulierung eines Ziels bezieht sich daher auf die Zusicherung gewünschter Eigenschaften; sie ist eine optative Aussage (eine Aussage in Form eines Wunsches) im Gegensatz zu einer indizierten Aussage (einer Aussage in Form eines Erfordernisses), und beschränkt durch die Rolle des Entwicklungsgegenstandes."

Motivation für die Verwendung zielorientierter Methoden Verschiedene Aspekte können durch die Zielorientierung im Rahmen der Erhebung von Anforderungen verbessert werden [vL01, Poh08]. Darunter fallen:

- *Vollständigkeit von Anforderungen*: Durch ein Zielmodell liegt ein präzise definiertes Kriterium vor, welches Rückschlüsse über die Vollständigkeit von Anforderungen liefert. Dabei kann sichergestellt werden, dass eine Anforderungsspezifikation in Bezug auf ein definiertes Zielmodell vollständig ist.
- *Angemessenheit von Anforderungen*: Ein Ziel kann erklären, warum eine Anforderung formuliert wurde. Dadurch kann die Existenz und Ausprägung einer Anforderung begründet werden. Des Weiteren können Ziele helfen, Anforderungen mit verschiedenen Stakeholdern zu diskutieren.
- *Konfliktlösung zwischen verschiedenen Perspektiven*: Ziele bilden die Basis, um Konflikte zu erkennen und diese gegebenenfalls zu lösen.
- *Trennung von stabilen und instabilen Informationen*: Generell sind Ziele wesentlich stabiler als technische Anforderungen für ein spezifisches Produkt, gerade wenn es sich um betriebswirtschaftliche Ziele handelt. Somit kann ein bestimmter Anteil eines Zielmodells bei der Entwicklung verschiedener Produkte wiederverwendet werden.
- *Beschreibung von mehr als nur Technik*: Ein Ziel kann zu einer Anforderung an ein zu entwickelndes System verfeinert werden. Es kann jedoch auch zu einer Annahme oder Erwartung an die Umwelt des zu entwickelnden Systems verfeinert werden. Somit kann auch begründet werden, warum bestimmte Anforderungen an ein technisches System *nicht* formuliert werden müssen. Dieses ist dann der Fall, wenn bestimmte Erwartungen an die Fähigkeiten eines Menschen (Operator, Autofahrer, etc.) gestellt werden, die dann zusammen mit dem technischen System zu geeigneten Eigenschaften und Verhalten des Gesamtsystems führen.

KAOS Die *Knowledge Acquisition in autOmated Specification* (KAOS) Methode entstand 1990 aus einer Kooperation der Universität von Oregon und der Universität von Louvain in Belgien. Letztere entwickelt die Methode bis heute weiter [Res07]. Dabei existiert neben der Methode

auch ein Beschreibungsmittel (das KAOS Metamodell, siehe Abschnitt 3.3.2) und ein Werkzeug (Objectiver, siehe Abschnitt 3.4). Nach [Hei05] ist KAOS besonders gut für die Dokumentation bzw. Spezifikation von Anforderungen geeignet.

GOMS Die *Goals, Operators, Methods and Selection rules* (GOMS) Methoden wurden ursprünglich von [CMN83] erwähnt. Diese kommen aus der Psychologie, insbesondere aus der Interaktion von Menschen und Computern. Sie können als Instrument in der Aufgabenanalyse verwendet werden, welche das Ziel verfolgt, die Aufgaben eines menschlichen Operators zu identifizieren und zu strukturieren. GOMS versucht dabei, die Ziele eines Nutzers zu formulieren, diese in Teilziele zu dekomponieren und zu zeigen, wie diese durch eine Interaktion erreicht werden.

Im Gegensatz zu anderen Methoden im Bereich der Aufgabenanalyse ermöglicht GOMS eine frühzeitige Überprüfung von Interaktionskonzepten. Dieses wird dadurch möglich, dass die elementaren Aufgaben eines Nutzers zusammen mit ihren Ausführungszeiten modelliert werden. Somit kann eine quantitative Evaluation der menschlichen Arbeit erfolgen, was wiederum den Vergleich verschiedener Entwurfsalternativen ermöglicht. Einige GOMS Methoden ermöglichen darüber hinaus eine Abschätzung der Einarbeitungszeit in einen neuen Arbeitsprozess.

Die GOMS Familie [JK96] umfasst dabei vier wesentliche Methoden: das ursprüngliche *Card-Morgan-Newell* (CMN) GOMS [CMN83], *Keystroke-Level-Model* (KLM) GOMS [CMN83], *Natural GOMS Language* (NGOMSL) [Kie88] und *Cognitive-Perceptual-Motor* (CPM) GOMS [GJA92]. Weitere Methoden im Rahmen der Aufgabenanalyse sind beispielsweise die *Hierarchical Task Analysis* (HTA), *Verbal Protocol Analysis* (VPA), *Task Decomposition*, *Sub-Goal Template Method* (SGT) und *Tabular Task Analysis* (TTA). Nach [Hei05] ist GOMS insbesondere für die Gewinnung von Anforderungen geeignet. Bei [Ant97] wird GOMS explizit im Kontext zielorientierter Methoden und Interdisziplinarität erwähnt.

GSN Die *Goal Structuring Notation* (GSN) wurde mit dem Fokus auf der Unterstützung von *Safety Cases* an der Universität York entwickelt. Im Speziellen soll die GSN, eine grafische Notation zur Argumentation, die Lücke zwischen Sicherheitsanforderungen und assoziierten Nachweisen schließen. Dafür werden unter Verwendung einer Zielnotation Sicherheitsziele zu Teilzielen und Lösungen dekomponiert. Diese Dekomposition umfasst die Notation von Strategien, welches die Argumentation unterstützt.

3.3 Beschreibungsmittel

Die eben beschriebenen Methoden nutzen eigene Modelle bzw. Beschreibungsmittel, die im Weiteren beschrieben werden. Solch eine formalisierte Definition von *Domain-Specific Languages* (DSL), also domänenspezifischer Sprachen, ist die Grundlage für den Einsatz modellgetriebener Entwicklung, dem *Model-Driven Development* (MDD). Somit wird es für einen Domänenexperten möglich, Modelle in seiner eigenen Sprache zu formulieren, die dann in andere Modelle oder Quellcode überführt werden können. Bei der Übersetzung in Quellcode können so abstrakte Modelle ausführbar und damit evaluierbar gemacht werden. Die Transformation in andere Modelle ist eine wichtige Grundlage für einen interdisziplinären Entwurf.

3.3.1 Domänenmodelle

Zur Strukturierung und Formalisierung von Domänenwissen existieren verschiedene Ansätze, die sich vor allem durch ihre formale Ausdrucksstärke unterscheiden (siehe Abbildung 3-2). Ein *Glossar* beschreibt lediglich Begriffe, ohne Relationen zwischen Begriffen. *Taxonomien* können ausschließlich monohierarchische Begriffsstrukturen (Einfachvererbung) abbilden. Ein *Thesaurus* verfügt darüber hinaus über zwei feste Relationen: die *Ähnlichkeit* und die *Synonymie*. Ein

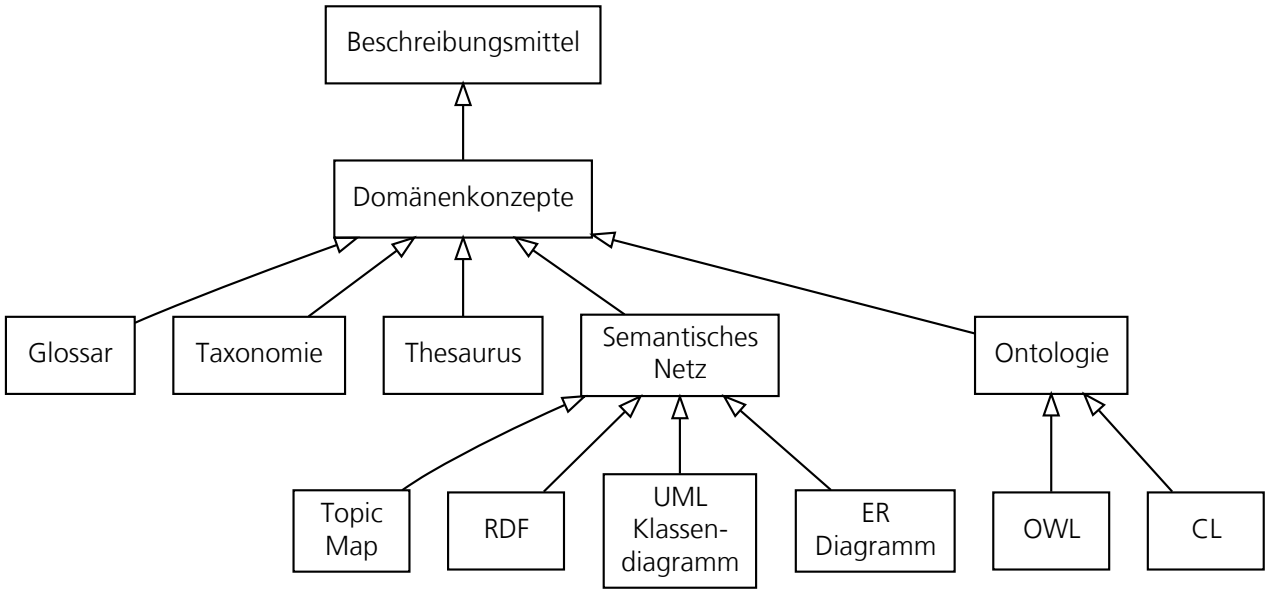

Abbildung 3-2: Eine Auswahl an Beschreibungsmitteln, die die Konzepte und Zusammenhänge einer Domäne beschreiben können.

weiteres Mittel sind *semantische Netze*, welche neben Begriffen beliebige gerichtete Relationen beschreiben können. Eine *Ontologie* umfasst neben Konzepten (Begriffen) und Relationen eine Menge von Regeln zur automatisierten Schlussfolgerung (Inferenzregeln). Für die genannten Beschreibungsmittel existieren verschiedene Spezialisierungen mit technischer Realisierung. Ein klassisches Beschreibungsmittel aus dem Bereich der Informationssysteme sind *Entity Relationship* (ER) Modelle [Che76]. Die *entities* stellen Konzepte einer Domäne dar, *relationships* beschreiben die Beziehungen zwischen den Konzepten. ER Diagramme verfügen über eine grafische Syntax. Ähnlich wie ER Modelle sind *Klassendiagramme* [OMG09c] in der UML eine Möglichkeit, eine grafische Notation zur Datenmodellierung. Die besondere Stärke von UML Diagrammen liegt in der guten Verständlichkeit der grafischen Notation. Des Weiteren sind UML Diagramme relativ einfach zu erstellen. Somit eignen sich UML Diagramme insbesondere zur *informellen* Modellierung und als Grundlage zur Diskussion von Konzepten einer Domäne.

Semantische Netze können mit Hilfe des *Resource Description Framework* (RDF) [MM04] beschrieben werden. Auch *Topic Maps* [ISO03b] eignen sich hierfür. Ontologien können insbesondere mit der *Web Ontology Language* (OWL) [DS04] modelliert werden, da sie die Modellierung von Inferenzregeln erlaubt.

Die *Object Management Group* (OMG) hat das sogenannte *Ontology Definition Metamodel* (ODM) [OMG09b], standardisiert. Dieses setzt eine Vielzahl der genannten Beschreibungsmittel in Bezug zueinander und zeigt auf, wie und in welchem Maße sich die Ausdrucksmöglichkeiten der genannten Mittel ineinander überführen lassen. Dazu gehören ER Diagramme, *Common Logics* (CL), RDF, OWL und XML Topic Maps.

OWL: Web Ontology Language

Die OWL ist ein offener Standard [DS04] vom W3C (World Wide Web Consortium). Inzwischen liegt die OWL in der zweiten Version vor [PPS08]. Die OWL 2 umfasst die Profile *OWL 2 EL*, *OWL 2 QL* und *OWL 2 RL* [MGH+09]. Diese beschreiben Teilmengen der OWL, um die Effizienz der Ausführung logischer Deduktion (Inferenz) zu verbessern, dabei wird jedoch die Ausdrucksstärke reduziert. OWL 2 EL ist insbesondere geeignet für Ontologien mit sehr vielen Klassen und Eigenschaften. Die Abkürzung steht für die dem Profil zugrunde liegenden Logikfamilie (EL [BBL05]). Die OWL 2 *Query Language* (QL) ist vor allem für Ontologien mit sehr vielen Individuen geeignet. Die Abkürzung ist darauf zurückzuführen, dass Anfragen auf standardisierte relationale

Abfragesprachen abgebildet werden können. Die OWL 2 *Rule Language* (RL) ist die mächtigste Teilsprache. Ihr Akronym lässt sich darauf zurückführen, dass die Inferenz sich mit standardisierten Regelbasen abbilden lässt.

Die OWL 2 erweitert die Semantik [Sch09a] von RDF. Weiterhin hat sie eine direkte Semantik im Sinne einer Beschreibungslogik [MPSG09], einer Prädikatenlogik erster Ordnung. Konkret handelt es sich um *SROIQ* [HKS06]. Die Zusammensetzung dieser Abkürzung sowie Syntax und Semantik von SROIQ sind in Abschnitt B.3 ausführlich dargestellt. Der Vorteil der direkten Semantik auf Basis einer Beschreibungslogik ist, dass in OWL beschriebene Ontologien einer formalen Betrachtungsweise zugeführt werden können. So kann etwa das Modell auf Konsistenz geprüft werden. Weiterhin ist logisches Schlussfolgern (Deduktion, Inferenz) möglich. Dieses erfolgt unter Verwendung sogenannter (semantischer) *Reasoner*.

Der Nachteil ist, dass die formale Modellierung mit der OWL mehr Lernaufwand als etwa UML Klassendiagramme erfordert. Des Weiteren verfügt die OWL über keine standardisierte grafische Notation.

Trotz der formalen Basis ist die OWL offen, begründet durch die *Open World Assumption* (OWA). Diese sagt aus, dass der Wahrheitsgehalt einer Aussage unabhängig davon ist, ob ein einzelner Beobachter sie als wahr erkennt. Das bedeutet, dass keine Schlussfolgerungen möglich sind, die sich auf *implizite* Annahmen stützen. Dieses macht die OWL sehr gut geeignet für den Einsatz in einem offenen, interdisziplinären Umfeld, wobei einzelne Entwickler nur einen Teil des Lösungsraums kennen.

Die für die OWL genutzte Beschreibungslogik ist eine entscheidbare Untermenge der Prädikatenlogik erster Stufe. Dieses ermöglicht es, aus vorhandenem Wissen neues Wissen zu erschließen. Die wesentlichen Elemente einer Beschreibungslogik sind Konzepte C, Rollen R, und Individuen c. Dabei handelt es sich bei den Individuen c um Instanzen der Konzepte C, während die Rollen R Beziehungen zwischen den Individuen c beschreiben. Temporale und modale Logik (ebenfalls eine Erweiterung der Prädikatenlogik) erlauben es, zeitliche Relationen auszudrücken. Solche Logiken sind im akademischen Bereich bereits erfolgreich mit Beschreibungslogik gekoppelt worden [AF98, AFC+01].

Weiterhin unterscheidet man unterschiedliche Formen von Beschreibungslogiken. Diese werden oft durch Buchstabenkombinationen [HPS03] repräsentiert. Die Beschreibungslogik SROIQ lässt sich in drei Teilen beschreiben:

- *Terminological Box* (Tbox): Die Tbox enthält dabei das axiomatische Wissen über die Konzepte einer Domäne, das terminologische Wissen.
- *Role Box* (Rbox): Die Rbox sammelt das axiomatische Wissen in Bezug auf Rollen.
- *Assertional Box* (Abox): Die Abox hingegen enthält das Fakten über Instanzen dieser Konzepte, sowie deren Beziehungen untereinander und repräsentiert somit den Zustand der modellierten Welt.

Im Weiteren wird die OWL Variante OWL 2 verwendet, welche auf der Beschreibungslogik *SROIQ* beruht und entscheidbar ist. Dieses Beschreibungsmittel spielt eine zentrale Rolle in dieser Arbeit. Die OWL wird mittels eines ebenfalls standardisierten *eXtensible Markup Language* (XML) [BPSM+06] Schemas oder als Erweiterung von RDF [MM04] serialisiert. Somit liegt eine einfache Interoperabilität für eine Werkzeugintegration vor.

SUMO Auf der Basis von OWL liegen eine Vielzahl von Ontologien vor, die unterschiedlichste Domänen beschreiben. Elementar sind sogenannte *Upper Ontologies*, die sehr generische Konzepte beschreiben und somit eine gemeinsame semantische Basis für Interoperabilität zwischen verschiedenen konkreten Ontologien ermöglichen. Bei [MCR07] findet sich ein

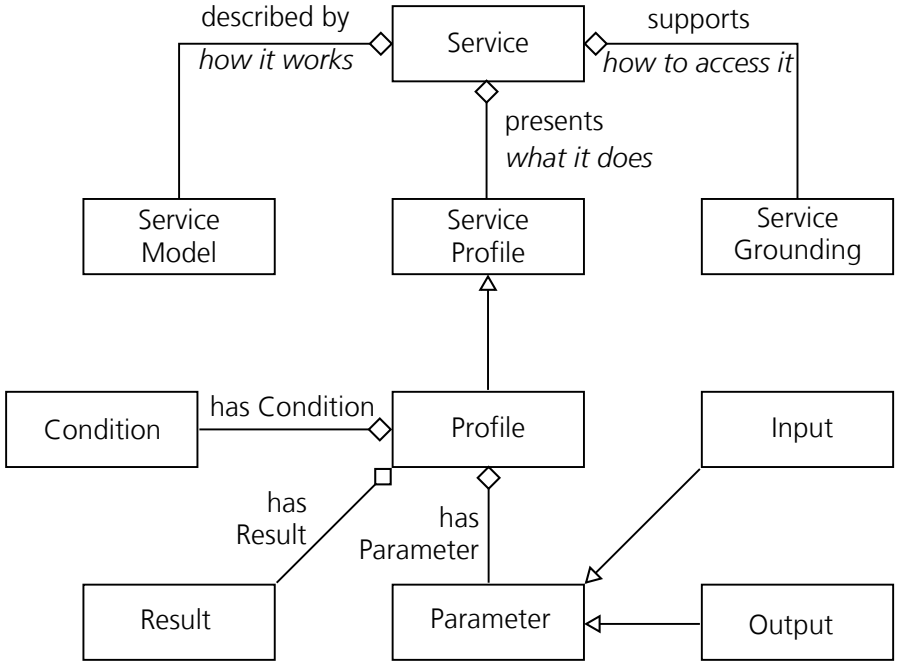

Abbildung 3-3: Die OWL-S stellt eine *Upper Ontology* für Dienste zur Verfügung.

Vergleich solcher generischer Ontologien. Dabei kann die *Suggested Upper Merged Ontology* (SUMO) [NP01] als einzige komplett auf *WordNet*, eine signifikante Sammlung englischsprachigen Wortschatzes, bezogen werden. SUMO wurde von einer IEEE Arbeitsgruppe[1] erarbeitet. Diese stellt weiterhin die *MId-Level Ontology* (MILO) sowie eine Reihe von Domänenontologien zur Verfügung, unter anderem auch eine für den Bereich Verkehr. Aus diesem Grunde stellt SUMO die Basis der in dieser Arbeit und im Projekt DeSCAS beschriebenen Domänenontologien, wie ausführlicher in Abschnitt 5.3 beschrieben. Aktuell umfasst SUMO 20.000 Begriffe und 70.000 dazugehörige Axiome[2].

OWL-S Bei OWL-S [W3C04] vom W3C handelt es sich um eine spezielle Upper Ontology, die den Bereich von Diensten im Internet fokussiert. Die OWL-S umfasst dabei sehr konkrete Teile, die sich explizit auf eine Dienstespezifikation in der *Web Services Description Language* (WSDL) [CCMW01] beziehen. Darüber hinaus umfasst sie eine abstrahierte Sicht auf den Dienst. Das macht es möglich, in WSDL spezifizierte Dienste sehr einfach mit anderen Domänen und Modellen zu verknüpfen.

Somit bietet eine solche Ontologie eine gute Basis für eine Verknüpfung von technischen Implementierungen mit Artefakten aus anderen Domänen und wird daher in Abschnitt 7 aufgegriffen. Das Metamodell ist in Abbildung 3-3 dargestellt. Dabei gibt das *Service Grounding* an, wie der Dienst technisch einzubinden ist. Das *Service Model* beschreibt das innere Verhalten des Dienstes (als Prozess) und das *Service Profile* beschreibt das äußere Verhalten des Dienstes. Dieses umfasst unter anderem dessen Voraussetzungen (*Condition*) sowie dessen Ergebnis (*Result*). Über *Parameter* (*Input* und *Output*) sind die Schnittstellen des Dienstes beschrieben.

3.3.2 Anforderungsmodelle

Zur Beschreibung von Anforderungen existieren eine Vielzahl von Beschreibungsmitteln (siehe Abbildung 3-4). Die Wahl eines Beschreibungsmittels wird auch hier vor allem durch die Anwendungsdomäne bestimmt. Werden Kundenanforderungen erhoben, so kann etwa ein UML *Anwendungsfalldiagramm* sinnvoll sein. Werden Architekturanforderungen erhoben, so

[1] IEEE P1600.1 Standard Upper Ontology Working Group (SUO WG): `http://suo.ieee.org/`
[2] Quelle 07.04.2010: `http://www.ontologyportal.org/`

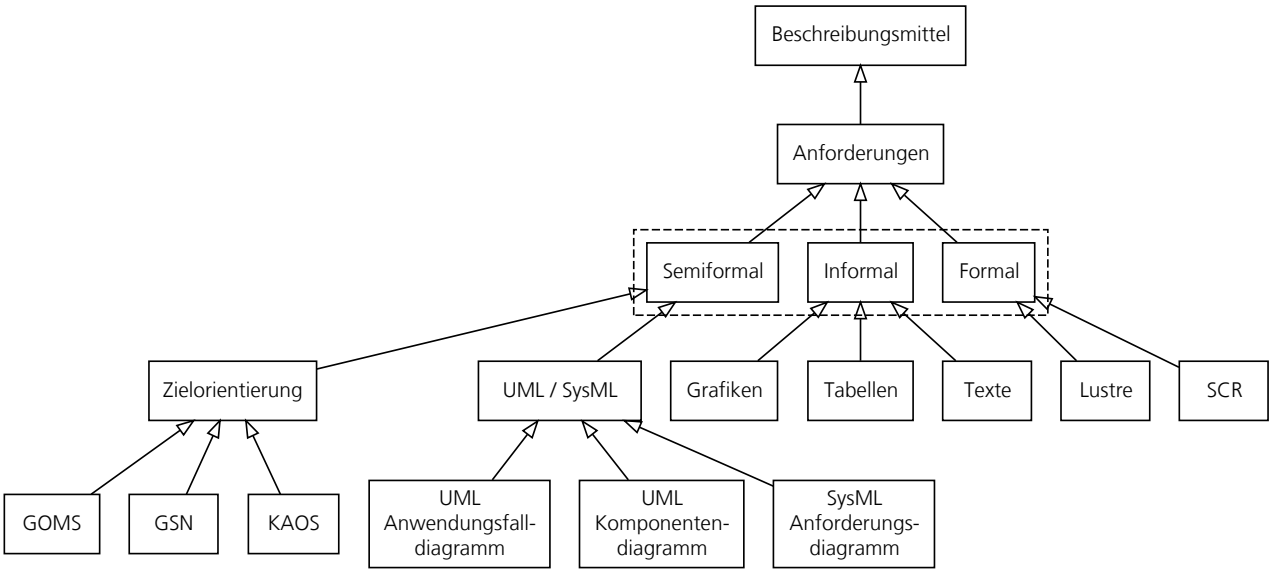

Abbildung 3-4: Eine Auswahl an Beschreibungsmitteln für Anforderungen.

kann ein *Komponentendiagramm* verwendet werden. Bei der vorliegenden Arbeit steht die Verwebung verschiedener Disziplinen im Vordergrund. Daher sind die Aspekte der Integration, der Verknüpfung und *Traceability* von Anforderungen von zentraler Bedeutung. Dieses umfasst etwa die Verfeinerung oder Ableitung neuer Anforderungen. Solche *Traceability Links* sind in der *Systems Modeling Language* (SysML) beschrieben. Die SysML ist wie die UML von der OMG standardisiert [OMG07] und verfügt über eine große Schnittmenge mit der UML. Bemerkenswert ist vor allem das *Anforderungsdiagramm*. Die SysML ist ebenfalls ein semiformales Beschreibungsmittel. Bei den formalen Mitteln ist vor allem *Lustre* hervorzuheben, eine synchrone deklarative Programmiersprache zur Beschreibung reaktiver Echtzeitsysteme. Für Lustre liegt mit *SCADE* ein Werkzeug vor, welches über einen zertifizierten Codegenerator verfügt. Dieses verbessert die Nutzung für sicherheitskritische Systeme erheblich. Daneben ist die *Software Cost Reduction* (SCR) Methode eine tabellarische Notation, die speziell für die formale Spezifikation von Anforderungen entwickelt wurde. Eine ausführliche, weiterführende Übersicht zu Beschreibungsmitteln für Anforderungen findet sich bei [Pai10a].

Zielorientierung

KAOS (Knowledge Acquisition in Automated Specification) Das KAOS Metamodell umfasst vier Basismodelle (*Zielmodellierung*, *Verantwortlichkeitsmodellierung*, *Operationsmodellierung* und *Objektmodellierung*). Die Modelle überschneiden sich an verschiedenen Stellen, indem einige Konzepte in mehreren Modellen verwendet werden können und somit die Basismodelle untereinander verknüpfen. Ein etwas vereinfachtes Metamodell zeigt Abbildung 3-5.

Die *Zielmodellierung* hat gewünschte Systemeigenschaften zum Inhalt und fokussiert das zu entwickelnde System. Der vorherrschende Anforderungstyp sind Systemanforderungen. Ein *Ziel* (Goal) kann durch weitere Ziele verfeinert werden, aber auch durch Domäneneigenschaften, Erwartungen oder Anforderungen, die dem Ziel dienen. Im Sinne der KAOS Methode klärt eine Verfeinerung, *wie* ein bestimmtes Ziel erreicht werden kann. In umgekehrter Richtung erklärt ein abstrakteres Ziel, *warum* ein Ziel formuliert wurde. Es darf Konflikte zwischen Zielen geben, wenn diese explizit formuliert werden. Eine *Anforderung* (Requirement) ist ein Ziel und wird von einem abstrakteren Ziel abgeleitet. Eine Anforderung wird einem Software-Agenten zugewiesen, der für die Erfüllung der Anforderung verantwortlich ist. Somit werden die Anforderungen für eine Software festgelegt. Auch eine *Erwartung* (Expectation) ist ein Ziel und erfüllt eine ähnliche Rolle wie die Anforderung. Im Gegensatz zu dieser kann eine Erwartung jedoch nicht eingefordert werden. Sie repräsentiert eine Annahme über die Umwelt des zu entwickelnden

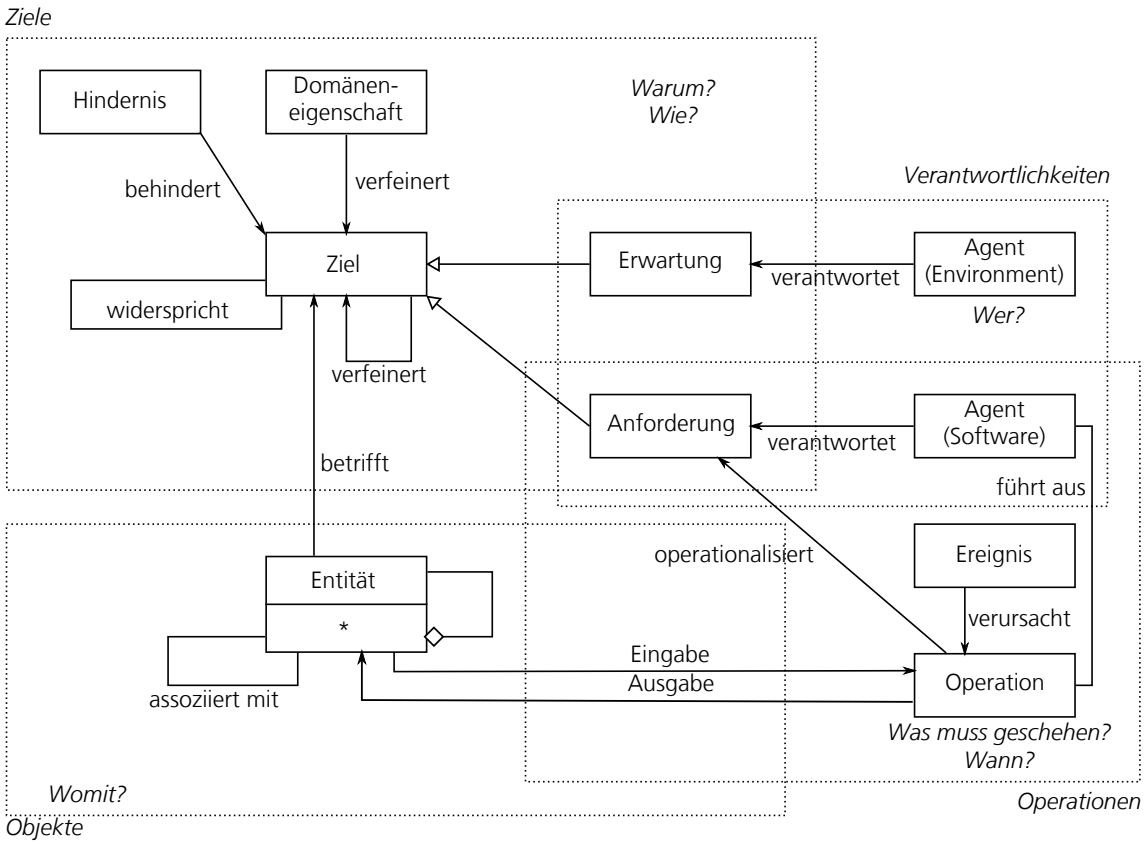

Abbildung 3-5: Das Metamodell von KAOS unterscheidet vier verschiedene Basismodelle, wobei insbesondere die Differenzierung zwischen Zielen und Anforderungen für zielorientierte Methoden wichtig ist.

Systems. Wie Anforderungen werden Erwartungen mit einem Agenten assoziiert, so kann die Verantwortung eines Operators (z.B. eines Fahrers oder Piloten) ausgedrückt werden. Eine *Domäneneigenschaft* (Domain Property) ist eine deskriptive Zusicherung im Kontext des zu entwickelndes Systems. Es kann sich um eine Domäneninvarianz (eine Eigenschaft, welche in der Domäne immer zutrifft, wie etwa die Wirkung eines Gesetzes oder Standards) oder um eine Domänenhypothese (eine Annahme über das Verhalten eines Objekts in einer Domäne) handeln. Eine Domäneneigenschaft kann genutzt werden, um die Vollständigkeit einer Zielverfeinerung zu argumentieren. Ein *Hindernis* (Obstacle) kann verwendet werden, wenn ein Ziel, Anforderung oder Erwartung aufgrund eines unerwünschten Verhaltens nicht erfüllt werden kann. In diesem Fall blockiert ein solches Hindernis ein Ziel.

Die *Verantwortlichkeitsmodellierung* hat Agenten sowie ihre Verantwortung für Ziele zum Inhalt. Der Fokus liegt auf dem System und dessen Umwelt. Der primäre Anforderungstyp sind Anforderungen an die Systemarchitektur. Ein *Agent (Software)* repräsentiert einen Teil des zu entwickelnden Systems, welches für die Erfüllung bestimmter Anforderungen verantwortlich ist. Dazu führt ein solcher Agent bei bestimmten Ereignissen bestimmte Operationen aus. Somit kann beschrieben werden, *wer* für die Erfüllung einer Anforderung verantwortlich ist. Ein *Agent (Umwelt)* repräsentiert einen Teil der Umwelt des zu entwickelnden Systems, an welchen bestimmte Erwartungen gestellt werden. Ein solcher Agent kann etwa ein Autofahrer sein.

Die *Operationsmodellierung* beschreibt das Verhalten eines Agenten (Software) zur Erfüllung eines Ziels. Der Fokus liegt auf dem System, die Anforderungen sind Anforderungen an das Systemverhalten. Eine *Operation* wird durch einen Software-Agenten aufgrund seiner Verantwortung für eine Anforderung durchgeführt und wird durch ein Ereignis ausgelöst. Eine Operation repräsentiert somit die Operationalisierung einer Anforderung und soll klären, *wann was* getan werden soll. Die Schnittstelle einer Operation wird durch Entitäten modelliert,

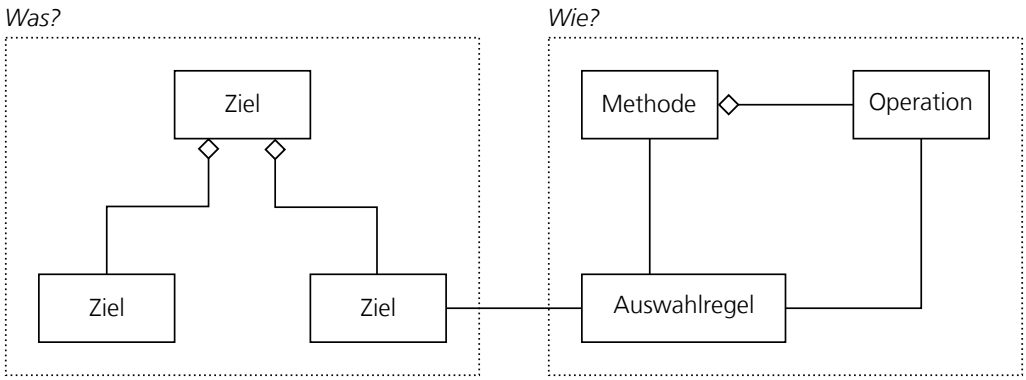

Abbildung 3-6: Das Metamodell von GOMS umfasst die Konzepte Ziel, Operation, Methode und Auswahlregel.

welche die Eingangs- und Ausgangsdaten einer Operation repräsentieren. Ein *Ereignis* löst eine bestimmte Operation aus.

Die *Objektmodellierung* hat ein konsistentes und komplettes Glossar zum Inhalt. Eine *Entität* (Entity) repräsentiert ein Konzept der Anwendungsdomäne. Sie kann zur Beschreibung der Schnittstelle einer Operation genutzt werden und weiterhin die Beschreibung eines Zieles anreichern, indem der Kontext eines Ziels beschrieben wird. Eine *Assoziation* (Association) beschreibt die Relation zwischen Entitäten.

Die wichtigsten Verknüpfungen in Bezug zur Nachvollziehbarkeit sind dabei die Folgenden. Eine *Verfeinerung* (Refinement) bedeutet, dass ein Ziel in weitere Ziele, Anforderungen oder Erwartungen verfeinert wird. Somit können verschiedene Abstraktionsebenen und Begründungen für Anforderungen dargestellt werden. Die *Operationalisierung* (Operationalisation) verknüpft die Zielmodellierung mit der Operationsmodellierung. Ähnlich wie eine Verfeinerung ermöglicht die Operationalisierung eine Form von Verfeinerung und Begründung (für ein spezifisches Verhalten). Die *Verantwortlichkeit* (Responsibility) für Erwartungen und Anforderungen können mit einem Agenten verknüpft werden, der dann für die Erfüllung der Erwartung bzw. Anforderung verantwortlich ist.

GOMS (Goals, Operators, Methods and Selection rules) Die basalen Modellelemente von GOMS sind in Abbildung 3-6 grafisch dargestellt. Ein *Ziel* (Goal) beschreibt, was ein menschlicher Bediener erreichen möchte. Ziele können zu Teilzielen dekomponiert werden. Eine *Operation* (Operator) steht im Dienste eines Ziels. Eine Operation kann dabei eine beobachtbare Handlung oder eine mentale Operation sein. *Methoden* (Methods) sind geordnete Tupel aus Operationen, die zur Erreichung eines Ziels kombiniert werden. Schließlich beschreiben *Auswahlregeln* (Selection rules), warum sich ein Nutzer für eine bestimmte Methode zur Erreichung eines Ziels entscheidet, wenn mehrere Methoden möglich sind.

GSN (Goal Structuring Notation) Die folgenden Konzepte sind die basalen Elemente der GSN [KW04], wie auch in Abbildung 3-7 dargestellt. Ein *Ziel* (Goal) ist normalerweise sicherheitsrelevant und repräsentiert eine Anforderung. Ziele können zu Teilzielen und Strategien dekomponiert werden. Schließlich sollte ein konkretes Ziel zu einer *Lösung* (Solution) verfeinert werden. Ein nicht entwickeltes Ziel repräsentiert offene Punkte. Die Rolle einer Lösung ist die Beschreibung davon, wie ein Ziel erreicht werden kann. Eine Lösung wird nicht weiter dekomponiert. Sie erläutert einen Weg, wie ein abstrakteres Ziel direkt erfüllt werden kann. Eine *Strategie* (Strategy) kann Teil der Zieldekomposition sein und adressiert dabei die Argumente, die zur Erfüllung eines abstrakteren Zieles gewählt werden. Dabei ergeben sich aus einer Strategie weitere Teilziele. Der *Kontext* (Context) beschreibt das Umfeld eines Ziels, einer Lösung oder einer Strategie.

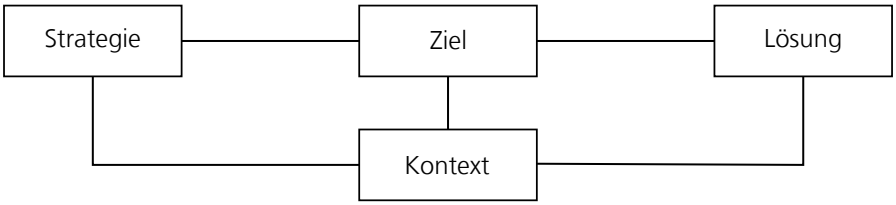

Abbildung 3-7: Das vereinfachte Metamodell der GSN.

3.4 Werkzeugunterstützung

Für einen effizienten Einsatz der genannten Methoden und Beschreibungsmittel existiert eine Reihe von Werkzeugen. Für die Modellierung von Domänenwissen in Form von Ontologien kann *Protégé* (siehe unten) eingesetzt werden. Kommerzielle Werkzeuge wie *Artisan Studio*, *Enterprise Architect* oder *Rhapsody* unterstützen die Modellierung von UML Modellen, die neben der Domänenmodellierung auch zur Formulierung von Anforderungen genutzt werden können. Auch wenn hierfür einfache Officeanwendungen nach wie vor sehr verbreitet sind, so werden im AAS Bereich verstärkt modellbasierte Werkzeuge genutzt. Neben den genannten UML Werkzeugen sind dies vor allem *MATLAB/SIMULINK*, *ASCET* und *SCADE*. SCADE ist aktuell das weltweit einzige Produkt, welches über einen zertifizierten (im Sinne der funktionalen Sicherheit) Codegenerator für sicherheitskritische Verkehrssysteme verfügt. Die zielorientierten Methoden verfügen ebenfalls über Werkzeugunterstützung, etwa *Objectiver* für KAOS, *CogTool* für GOMS oder *Artisan GSN Modeler* für GSN. Eine umfassende und ausführliche Übersicht, insbesondere zu Requirements Engineering Werkzeugen, findet sich bei [Pai10b, Pai10a].

Im Rahmen dieser Arbeit sind insbesondere *Protégé* und *Objectiver* von Bedeutung, da diese Werkzeuge für die Modellierung von Domänenwissen in Form von Ontologien bzw. Zielen und Anforderungen genutzt wurden.

Protégé

Protégé[3] wird unter der Führung der Stanford University als Open Source Projekt entwickelt und ist freie Software im Sinne der *Mozilla Public License* (MPL). Die aktuell vorliegende Version 4 der Software unterstützt die Bearbeitung von OWL 2 Modellen. Dabei lassen sich auch verschiedene Reasoner einbinden. Wesentliche Komponenten liegen in Form der freien Java Programmierschnittstelle *OWL API*[4] offen vor. Ähnliches gilt für den Reasoner *HermiT*[5], der an der University of Oxford entwickelt wird.

Objectiver

Objectiver[6] unterstützt die KAOS Methode und Beschreibungsmittel. Objectiver wird von Respect-IT[7] als kommerzielles Werkzeug entwickelt. In Zukunft soll neben der Modellierung von KAOS Modellen und dem Export von Anforderungsspezifikationen unter anderem die Einbindung von SAT Solvern für die Möglichkeiten einer formalen Verifikation geschaffen werden.

[3]The Protégé Ontology Editor and Knowledge Acquisition System: `http://protege.stanford.edu/`
[4]OWL API: `http://owlapi.sourceforge.net/`
[5]Hermit OWL Reasoner: `http://hermit-reasoner.com/`
[6]Objectiver: `http://www.objectiver.com/`
[7]Respect-IT: `http://www.respect-it.com/`

3.5 Zusammenfassung des Kapitels

Existierende Methoden im Bereich des *Domain Engineering* sind vor allem durch den Aspekt der Wiederverwendung motiviert. Im Kontext eines formalisierten, interdisziplinären Entwicklungsprozesses können diese jedoch für die Beschreibung einer Domäne dienen. Dieses wirkt sich positiv auf die Vollständigkeit und Verständlichkeit von Anforderungen aus. Bemerkenswert ist die Modellierung von Domänenwissen in Form offener, formaler Beschreibungslogik in OWL. Mit Protégé liegt für eine Modellierung mit der OWL ein geeignetes Werkzeug vor, welches auch im Rahmen dieser Arbeit verwendet wurde.

Im Bereich des *Requirements Engineering* wurde aufgezeigt, dass sich in Bezug zu den Forschungsfragen insbesondere zielorientierte Methoden eignen. Diese ermöglichen neben der Modellierung technischer Anforderungen auch die Modellierung nicht-technischer Anforderungen wie das erwartete Verhalten eines Autofahrers. Außerdem liegen im Fokus der Anwendungsdomäne mit KAOS, GOMS und GSN drei Methoden vor, die die Bereiche der *Funktionsentwicklung* (KAOS), *Faktor Mensch* (GOMS) und *Funktionale Sicherheit* (GSN) fokussieren. Sie verfügen allesamt über eigene Metamodelle, die als Beschreibungsmittel für eine Formalisierung genutzt werden können. Das Werkzeug Objectiver, welches die KAOS Methode umsetzt, wird im Rahmen dieser Arbeit verwendet.

Mit Blick auf die Verknüpfung formaler, konzeptioneller Modelle mit technischen Implementierungen existiert mit der OWL-S eine standardisierte Ontologie, welche explizit über eine technische Bindung zu Internetdiensten verfügt. Somit kann eine Konkretisierung formaler, konzeptueller Modelle (Ontologien) ermöglicht werden.

Mit Bezug zu den Forschungsfragen untermalt die formale Basis der OWL die Eignung als Beschreibungsmittel für die Formalisierung eines Vorgehensmodells und weiterer Analysemethoden. So liegen OWL Modelle nicht nur in einem offenen Standard vor, sie bringen über die Verknüpfung zu anderen Ontologien ihre Metamodelle immer explizit mit. Diese Offenheit bewirkt eine gute *Selbstbeschreibungsfähigkeit* der Modelle. Diese Selbstbeschreibungsfähigkeit macht Erweiterungen auf Metamodellebene möglich und erleichtert somit die flexible Verknüpfung von Modellen ungemein.

Dieser Grad Flexibilität und Offenheit fehlt vielen aktuellen Werkzeugen. Beim Werkzeug Objectiver etwa handelt es sich um ein kommerzielles Werkzeug für die zielorientierte Erhebung von Anforderungen. Diese Erhebung gelingt sehr gut, jedoch wäre eine größere Offenheit und Selbstbeschreibungsfähigkeit der Modelle wünschenswert, um Anforderungsartefakte einfacher zu verknüpfen. Ein Einbinden des Werkzeugs (wie in Kapitel 8 zu sehen) benötigt technische Expertise und entsprechend Zeit, um eine Modelltransformation zu entwickeln. Aktuelle Engineering Werkzeuge sollten von Beginn an auf Offenheit ausgelegt sein, denn diese wird zu einem immer wichtigeren Erfolgsfaktor.

3.6 Zusammenfassung des Teils: Grundlagen

Die in Kapitel 2 und Kapitel 3 beschriebenen Grundlagen werden im Laufe der vorliegenden Arbeit weiter aufgegriffen (siehe Abbildung 3-8).

Dabei findet auf Basis der beschriebenen domänenspezifischen Standards eine Definition der Disziplinen und wesentlicher Anforderungen innerhalb der AAS Entwicklung statt (Kapitel 4). Die Grundlagen zu den verschiedenen Prozessmodellen werden bei der Definition des verwobenen Vorgehensmodells genutzt (Kapitel 5). Die Grundlagen zu Beschreibungsmitteln, Methoden und Werkzeugen werden insbesondere bei der Modellverwebung sowie der Entwicklerunterstützung aufgegriffen (Kapitel 6).

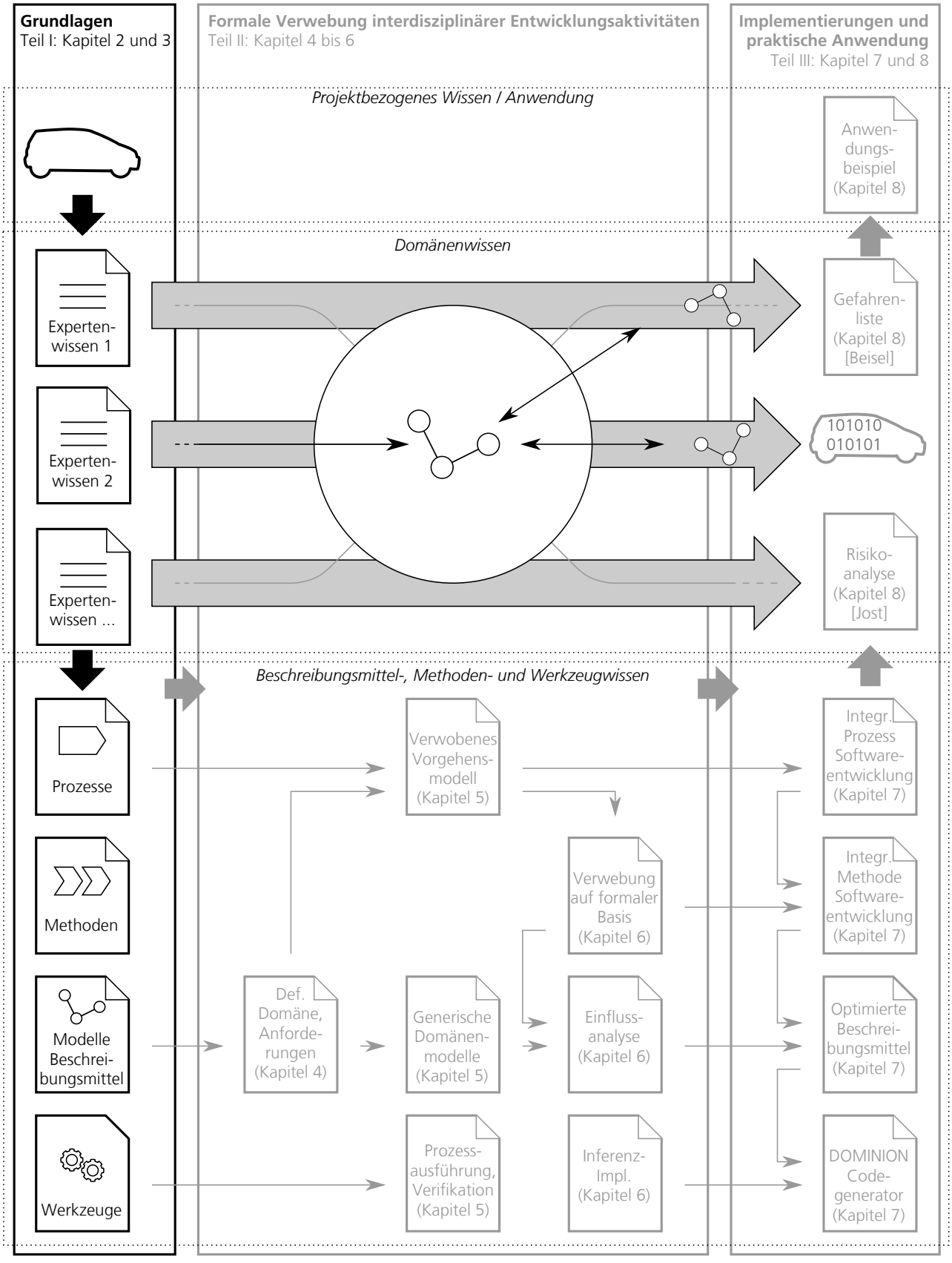

Abbildung 3-8: Struktur der Arbeit.

Weiterhin findet das vor allem in Kapitel 2 beschriebene Expertenwissen zur Entwicklung sicherheitskritischer Systeme Verwendung im Rahmen der exemplarischen Anwendung (Kapitel 8).

Teil II

Formale Verwebung interdisziplinärer Entwicklungsaktivitäten

4 Interdisziplinäre Systementwicklung

4.1 Ziele des Kapitels

Nach der Beschreibung der Grundlagen von Prozessen, Methoden, Beschreibungsmitteln und Werkzeugen wird in diesem Kapitel noch genauer auf die Definition der Interdisziplinarität (siehe auch [GJK+09]) eingegangen.

In Abschnitt 4.2 wird zunächst geklärt, was Interdisziplinarität im wissenschaftlichen Kontext bedeutet und wie sie sich von anderen Formen von Zusammenarbeit unterscheidet. Dazu wird eine Begriffsdefinition vorgenommen. Anhand wissenschaftstheoretischer Dimensionen wird der Begriff der Interdisziplinarität genauer betrachtet und strukturiert. In Abschnitt 4.3 findet eine Abgrenzung der AAS Entwicklung statt. Dazu wird ein Anwendungsbeispiel eingeführt. Dabei handelt es sich um ein ACC System, welches um einen Notbremsassistenten erweitert wird. Dieses dient für dieses und die folgenden Kapitel zur Veranschaulichung und Konkretisierung. Aus der Beschreibung der AAS Domäne und der Interdisziplinarität werden in Abschnitt 4.4 die Forschungsfragen weiter fokussiert und konkrete Anforderungen für die Kapitel 5 (Formalisierung des Entwicklungsprozesses) und Kapitel 6 (Verwebung interdisziplinärer Entwicklungsaktivitäten) formuliert.

4.2 Disziplinübergreifende Zusammenarbeit

Disziplinübergreifende Zusammenarbeit lässt sich anhand ihrer Form (siehe Abschnitt 4.2.1) charakterisieren und anhand von Dimensionen (siehe Abschnitt 4.2.2) beschreiben.

4.2.1 Formen disziplinübergreifender Zusammenarbeit

In der Wissenschaftstheorie ist die vorherrschende Meinung, dass bei der Entwicklung komplexer Produkte die Lösung nur durch die Zusammenarbeit von Spezialisten, nicht durch Allrounder allein erreicht werden kann. Es werden weiterhin verschiedene Arten von Zusammenarbeit beschrieben, deren Definition jedoch nicht einheitlich ist. Die Interpretation der folgenden Begriffe für diese Arbeit orientiert sich insbesondere an [Beh04, Ben04].

- *Multidisziplinarität*: Beteiligte Gruppen arbeiten vor allem nebeneinander, ohne intensive Interaktion. Dieses ist eine sehr lose Form von Zusammenarbeit.
- *Interdisziplinarität*: Beteiligte Gruppen arbeiten insbesondere *parallel*, dabei wird das behandelte Produkt zeitgleich von verschiedenen Gruppen bearbeitet. Die Gruppen müssen sich hier synchronisieren. Es findet Informationstransfer statt, jedoch werden unterschiedliche, spezialisierte Arbeitsmethoden verwendet. Durch die vergleichsweise lose Kopplung eignet sich diese Form der Zusammenarbeit insbesondere für komplexe Systementwicklungen.
- *Transdisziplinarität*: Beteiligte Gruppen arbeiten vor allem *direkt* zusammen. Die Zusammenarbeit ist hier intensiver als bei der Interdisziplinarität. Bestimmte Leistungen können nur im transdisziplinären Kontext erbracht werden. Dieses beinhaltet etwa die Entwicklung neuer Methoden in einem fachlichen Grenzgebiet. Aufgrund der starken Abhängigkeiten skaliert diese Vorgehensweise jedoch nur bedingt für große Projekte.

Für die Entwicklung von Assistenz- und Automationssystemen im Automobil spielen dabei vor allem die Formen der *Interdisziplinarität* und *Transdisziplinarität* eine wichtige Rolle. Transdisziplinarität ist entsprechend ihrer Definition an Stellen anzuwenden, die entweder mehrere beteiligte Gruppen direkt betreffen (z.B. bei der Gestaltung der Schalter eines ACC Systems) oder sich eine Aufteilung von Aufgaben nicht ohne Weiteres vornehmen lässt (z.B. Untersuchungen zur Bedienbarkeit oder Ergonomie). Interdisziplinarität ist vor allem dort sinnvoll, wo explizit Anforderungen formuliert werden können (z.B. die minimale Geschwindigkeit für ein ACC System) und es weiterhin eine klare Aufteilung für die Bearbeitung der Aufgaben gibt.

Ein adäquates Prozessmodell muss somit im Kontext der Entwicklung von Assistenz- und Automationssystemen im Verkehr Aspekte der *Transdisziplinarität* und der *Interdisziplinarität* abdecken können, jedoch mit klarem Schwerpunkt im Bereich der *Interdisziplinarität*.

4.2.2 Dimensionen disziplinübergreifender Zusammenarbeit

Um einen strukturierten Blick auf die disziplinübergreifende Zusammenarbeit zu erlangen, werden die Dimensionen *Institution und Organisation*, *Zugang und Methoden*, *Gegenstände und Objektfelder* und *Bildung und Lernprozesse* aus der Wissenschaftstheorie[1] hinzugezogen.

1. *Institution und Organisation*: Ein Vorgehensmodell muss es ermöglichen, verschiedene Disziplinen zu integrieren bzw. deren Arbeit zu synchronisieren. Dazu gehört der kontinuierliche Austausch von Ideen, Konzepten, sowie Wissen und Erkenntnissen. Weiterhin muss das Prozessmodell so offen gestaltet sein, dass es in der Lage ist, neue Disziplinen aufzunehmen. Zum interdisziplinären kontinuierlichen Austausch bietet sich ein iteratives Vorgehen an.

2. *Zugang und Methoden*: Werden bei der Entwicklung keine Methoden ausgetauscht, so sollte es jedoch möglich sein, die Methoden kombiniert im Rahmen des Gesamtprozesses anzuwenden. Das bedeutet, dass sich die Methoden insbesondere bezüglich ihrer Ergebnisse kombinieren lassen müssen. Dazu gehört die Auswahl und Verwebung geeigneter Methoden aus den verschiedenen Disziplinen. Wenn neue Methoden entwickelt werden, sollte der überlagerte Prozessrahmen offen gestaltet sein, um diese neuen Methoden aufnehmen zu können.

3. *Wissen und Erkenntnis*: Domänenwissen muss einem Entwicklungsstrang zugeordnet werden können. Gleichzeitig muss es jedoch offen gestaltet sein, dass es transferiert und mit einer anderen Domäne verknüpft werden kann. Insbesondere für die Verknüpfung und den Wissenstransfer bietet sich eine Formalisierung des Domänenwissens an.

4. *Gegenstände und Objektfelder*: Objektfelder bilden die Grundlage, auf der Domänenwissen aufbaut. Hier werden die Begriffe beschrieben, die dann im Bereich des Wissens miteinander verknüpft werden. Auch hier ist neben einer Formalisierung von Begriffen, die die Grundlage für die Formalisierung von Wissen bildet, insbesondere die Offenheit für Verknüpfungen wichtig.

Tabelle 4-1 liefert eine strukturierte Zusammenfassung der Kernaspekte von *Interdisziplinarität* und *Transdisziplinarität* bezüglich der genannten Dimensionen. Sie wird im Weiteren als Grundlage für die Definition von Anforderungen zur interdisziplinären Systementwicklung in Abschnitt 4.4 genutzt.

[1] `http://www.itas.fzk.de/tatup/052/schm05a.htm`

Interdisziplinarität	Transdisziplinarität
Institution und Organisation	
Synchronisierung paralleler Entwicklungsstränge und Informationstransfer über das Vorgehensmodell	Vorgehensmodell ist offen für Erweiterungen durch neu entstehende Disziplinen
Zugang und Methoden	
Kompatibilität und Verwebung unterschiedlicher Methoden	Verschiedene Disziplinen nutzen gemeinsame methodische Basis
Wissen und Erkenntnis	
Spezialisierte Wissensbasis für jeden Entwicklungsstrang auf gemeinsamer, mit den anderen abgeglichenen Basis	Eine einzelne gemeinsame Wissensbasis
Gegenstände und Objektfelder	
Verknüpfen unterschiedlicher Objektfelder durch die Verwendung eines geeigneten Beschreibungsmittel für formalisiertes Wissen	Beschreibungsmittel für Domänenwissen handhaben gemeinsame und neue Objektfelder, leichte Erweiterbarkeit

Tabelle 4-1: Dimensionen von Trans- und Interdisziplinarität.

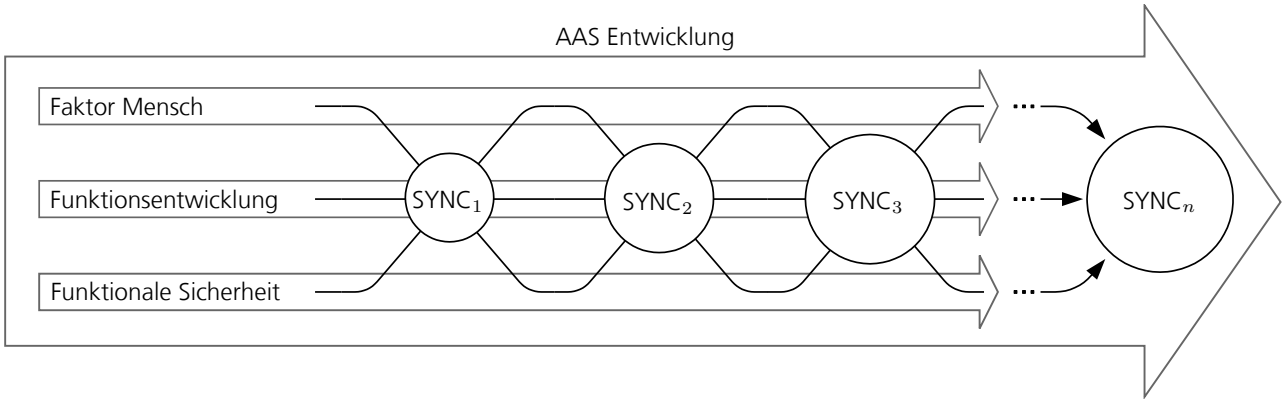

Abbildung 4-1: In der *AAS Entwicklung* dominieren die Entwicklungsstränge *Funktionsentwicklung*, *Funktionale Sicherheit* und *Faktor Mensch*.

4.3 Assistenz- und Automationssysteme

Im Weiteren werden drei zentrale Entwicklungsstränge [GJBK08] beschrieben. Dabei wird eine vor allem *interdisziplinäre* Sichtweise angenommen, da die Entwicklung von Assistenz- und Automationssystemen so komplex ist, dass Interdisziplinarität das bestimmende Paradigma ist. In Abbildung 4-1 sind die drei wichtigsten Entwicklungsstränge dargestellt. Dieses sind *Faktor Mensch*, *Funktionale Sicherheit* und *Funktionsentwicklung*. Diese Entwicklungsstränge stellen die interdisziplinäre Entwicklung im Rahmen des globalen Entwicklungsstrangs *AAS Entwicklung*.

Faktor Mensch

Aufgrund des steigenden Anteils automatisierter elektronischer Systeme im Automobil könnten Probleme, wie sie aus anderen Bereichen des Verkehrs bekannt sind (z.B. *mode confusion* aus der Luftfahrt [WNCF89]), auch im Bereich des Automobils auftreten. Aufgrund der Wiener

Konvention zum Straßenverkehr [UNE68] werden Automobile auch zukünftig vom Menschen gefahren oder zumindest überwacht werden.

Der Faktor Mensch ist besonders früh in der Entwicklung zu berücksichtigen. Das ist besonders im Kontext der Definition und Evaluation von Anforderungen wichtig. Die Evaluation erfolgt üblicherweise unter Verwendung technischer Prototypen, was wiederum ein iteratives Vorgehen motiviert. Die Ergebnisse der Evaluation sowie die damit verbundene Reflexion des Faktor Mensch führen zu Erkenntnissen, die in die Anforderungen an weitere Prototypen einfließen können. Die im Bereich des Faktor Mensch ausgeführten Versuche stellen hohe Anforderungen an die Verfügbarkeit und Flexibilität adäquater Test- und Evaluationsplattformen, inklusive Simulatoren und Testfahrzeuge. In diesem Entwicklungsstrang sind viele Psychologen beteiligt.

Funktionale Sicherheit

Durch die IEC 61508 und ISO DIS 26262 als deren Derivat wird der Nachweis risikosenkender Maßnahmen ein wichtiger Entwicklungsfaktor für Hersteller und Zulieferer im Automobilbereich. Elementarer Bestandteil des Vorgehens für die funktionale Sicherheit ist die Risikoanalyse, in der auch der Faktor Mensch eine wichtige Rolle einnimmt. Dabei ist vor allem die *Kontrollierbarkeit* von zentraler Bedeutung.

Funktionsentwicklung

Dieser Entwicklungsstrang stellt die technische Entwicklung des Systems, bestehend vor allem aus der Entwicklung von Funktionen und Architekturen. Dabei sind Entwickler in diesem Strang vor allem Ingenieure. Gerade dieser Strang ist in weitere Domänen und Entwicklungsstränge unterteilt (Hardware, Software, etc.). Der Entwicklungsstrang hat direkte Interaktion mit der funktionalen Sicherheit, da Sicherheitsanforderungen vornehmlich extrafunktionale Anforderungen an die Entwicklung stellen. Diese betreffen sowohl das Produkt als auch den Prozess. Dieses umfasst also das Vorgehen genauso wie konkrete Maßnahmen bei der Umsetzung einer Systemarchitektur in Hardware und Software. Erkenntnisse aus Evaluationen zum Faktor Mensch sind ebenfalls als funktionale und extrafunktionale Anforderungen in das technische System integriert.

Beispiel: ACC mit Notbremssystem

Im Weiteren wird der Entwurf eines Systems skizziert, das in den folgenden Teilen verwendet wird, um abstrakte Modellierungen verständlich und problembezogen darzustellen. Dabei steht neben der Beschreibung eines Assistenz- und Automationssystems die Beschreibung eines dazugehörigen Entwurfs im Vordergrund. Ausgehend von einem existierenden ACC System mit bekannter Systemarchitektur soll eine Erweiterung vorgenommen werden. Diese bringt bei drohender Gefahr eines Auffahrunfalls das Fahrzeug autonom zum Stehen, siehe auch Abbildung 4-2. Dabei verwendet das ACC bereits ein Radar, um das vordere Fahrzeugumfeld zu erfassen. Das ACC ist jedoch als Komfortanwendung ausgelegt und agiert im weiteren Abstand zum vorausfahrenden Fahrzeug. Das *Emergency Braking System* (EBS) hingegen kommt zum Einsatz, wenn nur noch ein geringer Abstand vorliegt und ein Auffahrunfall droht.

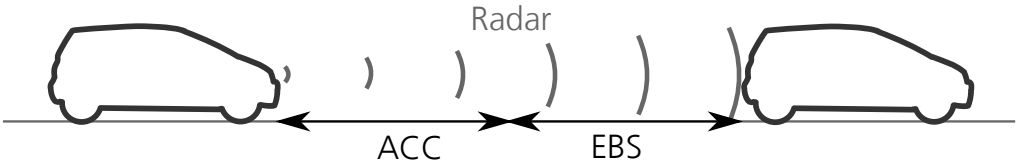

Abbildung 4-2: Das zusätzliche Notbremssystem EBS ergänzt das ACC.

Um ein solches System möglichst schnell auf den Markt zu bringen und Kosten zu sparen, soll dazu ein großer Teil des ACC Systems wiederverwendet werden. Ferner soll auch die Änderung an der Systemarchitektur gering gehalten werden, um den technischen Aufwand der Integration des EBS gering zu halten. Eine bereits vorhandene Komponente ist dabei der universelle Bremseingriff über das *Electronic Stability Program* (ESP). Darüber hinaus ist insbesondere der Umfeldsensor (Radar) bereits integriert. Der verfügt über eine Diagnosefunktion, die die Verfügbarkeit und Integrität der Sensorinformation prüft. Da diese Funktion auch beim ACC als sicherheitskritisch eingestuft wurde, könnte diese möglicherweise mit leichten Anpassungen wiederverwendet werden.

Neben der Frage nach einer adäquaten technischen Realisierung ergeben sich nun aus Sicht der Funktionsentwicklung bzw. des Architekturentwurfs Fragestellungen, die insbesondere die funktionale Sicherheit und den Faktor Mensch betreffen. Solche Fragestellungen umfassen etwa:

> „Ändert sich die Risikobereitschaft eines Autofahrers, wenn ihm bewusst ist, dass sein Fahrzeug über einen Notbremsassistenten verfügt? Wie gut ist eine fehlerhafte Auslösung des Systems kontrollierbar? Wie kann eine ausreichende Kontrollierbarkeit erreicht und nachgewiesen werden?"

Aus Sicht der Systemarchitektur mit Bezug zur Sicherheit ist vor allem der Einfluss der neuen Komponenten und ihrer Anforderungen wichtig:

> „Reichen die in der Entwicklung getroffenen Maßnahmen aus, um ein ausreichend sicheres Gesamtsystem zu konstruieren? Ändert sich die ASIL Klassifizierung? Kann das Gesamtsystem erfolgreich zugelassen werden? Was muss hierfür getan werden?"

Aspekte wie Zertifizierbarkeit, Kontrollierbarkeit und Sicherheit haben maßgeblichen Einfluss auf Entscheidungen im Bereich der Funktionsentwicklung und Architekturdefinition (weitere standardisierte Kriterien finden sich in [ISO01b]). Dabei handelt es sich insbesondere um extrafunktionale Aspekte, die Nebenbedingungen im Rahmen einer Optimierung des Gesamtentwurfs darstellen. Sie engen somit den Freiraum des Entwurfs ein, hier insbesondere den der Funktionsentwicklung.

An dieser Stelle kann außerdem die Bedeutung der Offenheit in Zusammenhang mit der Formalisierung veranschaulicht werden, dazu siehe auch Abbildung 4-3. Für den Entwickler stellt sich die Frage nach einer ASIL Bewertung des geplanten EBS. Gerade für ein neu zu entwickelndes System liegen noch viele Annahmen und dazu gehörende formalisierte Aussagen *implizit* vor. Folgen nun auf diesen impliziten Annahmen Schlussfolgerungen anhand einer geschlossenen Logik, so können diese unbekannte Nebenwirkungen mit sich bringen. Im Beispiel könnte somit eine Aussage zum ASIL für das EBS abgeleitet werden, obwohl gar keine Risikoanalyse erfolgt ist. Eine offene Logik liefert in diesem Fall keine Aussage, da nur Schlussfolgerungen anhand *expliziten* Wissens möglich sind. Das ist ein wesentlicher Vorteil der offenen Logik. Aber es ist auch ein erhöhter Modellierungsaufwand erforderlich, um implizite, aber notwendige Fakten explizit zu formulieren. Nutzt man die Modellierung jedoch in vielen Projekten (wie in diesem Beispiel für die Risikoanalyse), so lohnt sich der zusätzliche Aufwand schnell.

4.4 Anforderungen zur interdisziplinären Systementwicklung

Vor dem Hintergrund der Interdisziplinarität des Entwurfs von Assistenz- und Automationssystemen liegt eine besondere Herausforderung in der Tatsache, dass einzelne Entwickler nicht über ein ausreichendes Wissen verfügen (können), die Konsequenzen einzelner Entwurfsentscheidungen gegeneinander abzuwägen. An dieser Stelle kann eine Unterstützung dazu beitragen, ein neues System effizient und termingerecht fertigzustellen.

Wissensbasis
Für das *ACC* gibt es 10 *ASIL* Anforderungen.
Für das *Navigationssystem* gibt es 0 *ASIL* Anforderungen.

Frage
Wie viele *ASIL* Anforderungen gibt es für das *EBS*?

Offene Logik	**Geschlossene Logik**
Keine Antwort.	Antwortet, z.B.: 0.

Abbildung 4-3: Unterschied zwischen offener und geschlossener Logik im Kontext interdisziplinärer Entwicklung.

Hierfür werden Anforderungen abgeleitet, die insbesondere die Forschungsfragen bezüglich Formalisierung und Verwebung einzelner Entwicklungsaktivitäten aufgreifen. Dabei werden die Anforderungen den Kategorien *Vorgehensmodell* und *Wissensbasierte Verwebung* zugeordnet. Weiterhin werden die einzelnen Anforderungen kurz begründet. Die Anforderungen sind in Tabelle 4-2 zusammengefasst.

Dabei werden die Anforderungen an ein *Vorgehensmodell* (V1 - V9) weiter begründet. Diese Anforderungen geben einen Rahmen für das Vorgehensmodell in Kapitel 5 vor.

V1. *V-Modell*: Da es sich um die Entwicklung potentiell sicherheitskritischer Systeme handelt, sind Entwurfsentscheidungen zu verifizieren und zu validieren. Daher eignet sich eine Anlehnung an ein V-Modell. Des Weiteren ist der wichtigste Referenzprozess, das Vorgehensmodell der ISO DIS 26262, an ein V-Modell angelehnt.

V2. *Parallelisierung*: Im Gegensatz zur Transdisziplinarität ist im Sinne eines interdisziplinären Vorgehens wichtig, dass jeder Entwicklungsstrang seine in hohem Maße spezialisierten Werkzeuge und Methoden im jeweiligen Fachgebiet einsetzen kann. Dazu ist eine Parallelisierung von Entwicklungsaktivitäten notwendig.

V3. *Synchronisation*: Trotz eines weitgehend parallelisierten Vorgehens müssen Interaktionspunkte vorhanden sein, die einen Austausch zum Entwicklungsgegenstand und somit eine Synchronisation ermöglichen.

V4. *Iteratives Vorgehen*: Insbesondere vor dem Hintergrund menschzentrierter Entwicklung ist die Erstellung von Prototypen zur Evaluation und Bewertung unverzichtbar.

V5. *Granularität*: Falsches oder unzureichendes Verständnis von Anforderungen basiert häufig auf der Tatsache, dass Anforderungen in einer Granularität von Wissen ausgedrückt werden, die nicht alle Beteiligten verstehen. Dieses ist eine *produktbezogene* Granularität. Zum anderen existiert auch eine *prozessbezogene* Granularität insofern, als dass die jeweiligen Aktivitäten bei einem gemeinsamen Austausch häufig eine unterschiedliche Tiefe aufweisen.

V6. *Ausführbarkeit*: Die Ausführbarkeit eines Vorgehensmodells bedeutet insbesondere, dass den einzelnen Konzepten des Vorgehensmodells eine temporale Semantik zugewiesen wird. So kann eine Instanz des Vorgehensmodells auf eine Sprache zur Workflowunterstützung abgebildet werden.

V7. *Verifizierbarkeit*: Um Fehler wie im RESPONSE 3 CoP auszuschließen, müssen Eigenschaften wie die *Erreichbarkeit* (alle Phasen im konkreten Prozess können und werden erreicht) und

Symbol	Name	Anforderung
Vorgehensmodell		
V1	V-Modell	Das Vorgehensmodell muss die Modellierung eines V-Modells ermöglichen.
V2	Parallelisierung	Entwicklungsstränge müssen unabhängig voneinander arbeiten können.
V3	Synchronisation	Parallele Entwicklungsstränge müssen synchronisiert werden.
V4	Iteratives Vorgehen	Das Vorgehensmodell muss iterative Entwicklung unterstützen.
V5	Granularität	Das Vorgehensmodell muss verschiedene Ebenen von Granularität unterstützen.
V6	Ausführbarkeit	Das Vorgehensmodell muss ausführbar sein.
V7	Verifizierbarkeit	Auf einer konkreten Instanz des Vorgehensmodells müssen die Eigenschaften der Erreichbarkeit und Endlichkeit nachweisbar sein.
V8	Formalisierung	Das Vorgehensmodell muss formalisiert beschrieben werden.
V9	Validation	Das Vorgehensmodell muss die Prozessmodelle der ISO DIS 26262 und RESPONSE 3 CoP ausdrücken können.
Wissensbasierte Verwebung		
W1	Semantik	Anforderungen, die an andere Stränge im Entwicklungsprozess gerichtet sind, müssen bezüglich ihrer Bedeutung eindeutig und verständlich sein.
W2	Offenheit	Anforderungen müssen sich auf explizit formuliertes Wissen stützen, Schlussfolgerungen auf einer offenen Logik basieren.
W3	Formalisierung	Zur Formulierung von Anforderungen genutztes Domänenwissen muss offen und flexibel formalisiert werden.
W4	Kompatibilität	Methoden müssen kompatibel zu Methoden anderer Entwicklungsstränge sein.
W5	Konfliktlösung	Anforderungen an andere Entwicklungsstränge sind zu dokumentieren.
W6	Traceability	Es soll wissensbasierte Traceability (Nachvollziehbarkeit) gewährleistet sein.
W7	Handlungsplanung	Die Planung der Entwurfsentscheidungen eines einzelnen Entwicklers muss unterstützt werden.

Tabelle 4-2: Anforderungen an ein Vorgehensmodell und die wissensbasierte Verwebung.

Endlichkeit (die Entwicklung wird in endlicher Zeit durchlaufen) auf einer konkreten Instanz des Vorgehensmodells nachweisbar sein.

V8. *Formalisierung*: Um die Aspekte der *Ausführbarkeit* und *Verifizierbarkeit* abzusichern, ist eine Formalisierung des Vorgehensmodells notwendig.

V9. *Validation*: Dieses soll sicherstellen, dass ein nach dem zu definierenden Vorgehensmodell entwickeltes System zertifizierbar ist. Somit kann das Vorgehensmodell für die Entwicklung sicherheitskritischer Systeme eingesetzt werden. Weiterhin kann so geprüft werden, ob das Vorgehensmodell geeignet ist, Prozesse der AAS Domäne abzubilden.

Die zweite Kategorie von Anforderungen aus Tabelle 4-2 betrifft die *Wissensbasierte Verwebung* (W1 - W7). Diese werden in Kapitel 6 aufgegriffen.

W1. *Semantik*: Unvollständige oder inkonsistente Anforderungen sind der häufigste Fehler in der Produktentwicklung überhaupt.

W2. *Offenheit*: Im interdisziplinären Kontext ist die explizite Formulierung von Anforderungen wichtig, um Entscheidungen auf impliziter Basis zu vermeiden. Hierzu ist eine offene Logik zu verwenden.

W3. *Formalisierung*: Eine Formalisierung des Domänenwissens bietet sich an, um den Aspekt der *Semantik* zu stützen. Dieses bezieht sich insbesondere auf Vollständigkeit, Korrektheit, Verständlichkeit und Konsistenz von Anforderungen im Hinblick auf wohldefiniertes Domänenwissen. Weiterhin wird durch die Formalisierung der möglich werdende Einsatz generischer Werkzeuge unterstützt.

W4. *Kompatibilität*: Bei der Auswahl konkreter Methoden ist auf Kompatibilität zu Methoden anderer Disziplinen zu achten, um Entwicklungsaktivitäten stärker zu verweben.

W5. *Konfliktlösung*: In einer interdisziplinären Entwicklung ist mit Konflikten zu rechnen. Aus dem Stand der Technik (siehe Abschnitt 3.2.2) ist bekannt, dass verschiedene Perspektiven zur Konfliktlösung getrennt dokumentiert werden sollten.

W6. *Traceability*: Die Nachvollziehbarkeit ist das wichtige Element in der prozessbezogenen Zertifizierung eines sicherheitskritischen AAS. Um diese Traceability mit geringem Aufwand herstellen zu können, ist eine automatisierte Generierung von Dokumenten für die Zulassung wichtig.

W7. *Handlungsplanung*: Eine Unterstützung bei Entwurfsentscheidungen ist wichtig, um im Rahmen einer komplexen Gesamtentwicklung die Auswirkung einer Entwurfsentscheidung auf eigenes Handeln und das der anderen Entwicklungsstränge abschätzen und bewerten zu können.

4.5 Zusammenfassung des Kapitels

Die Begriffe *Interdisziplinarität*, *Transdisziplinarität* und *Multidisziplinarität* wurden für diese Arbeit eingeführt. Im Rahmen dieser Arbeit ist vor allem die Interdisziplinarität wichtig, bei dem ein konkretes AAS in der wiederholenden Interaktion zwischen den Entwicklungssträngen *Faktor Mensch*, *Funktionsentwicklung* und *Funktionale Sicherheit* entsteht. Dabei findet ein Austausch von Wissen statt, jedoch kein Austausch von Methoden.

Zur weiteren Konkretisierung wurde ein Anwendungsbeispiel eingeführt. Dabei handelt es sich um ein ACC System, welches um die Funktionalität eines Notbremsassistenzsystems erweitert werden soll. In Bezug auf die in der Einleitung beschriebenen Forschungsfragen eignet sich dieses Beispiel besonders deshalb, da hier bei einem existierenden System Anforderungen geändert

werden. Dieses hat Auswirkungen über die Funktionsentwicklung hinaus, insbesondere die funktionale Sicherheit betreffend.

Aus der konkretisierenden Definition von Interdisziplinarität wurden Anforderungen abgeleitet. Die Anforderungen richten sich dabei an das in Abschnitt 5 definierte Vorgehensmodell und an die wissensbasierte Modellverwebung in Abschnitt 6. Für das Vorgehensmodell ist vor allem dessen formale Basis wichtig, die automatisierte Analysen auf Prozessebene ermöglicht. Für die wissensbasierte Verwebung von Entwicklungsaktivitäten ist zentral, dass individuelle Entwickler ihre Entwurfsentscheidungen im Gesamtkontext der Entwicklung von AAS reflektieren können. Hierfür ist die Möglichkeit von logischem Schlussfolgern bezüglich des modellierten Domänenwissens eine wichtige Anforderung. Somit verfeinert die Anforderungsdefinition die Forschungsfragen (*Formalisierung des Entwicklungsprozesses* und *Verwebung interdisziplinärer Entwicklungsaktivitäten*).

5 Interdisziplinäres Vorgehensmodell und formales Rahmenwerk

5.1 Ziele des Kapitels

Basierend auf den Anforderungen, die im vorangegangen Kapitel 4 formuliert wurden, wird in diesem Kapitel ein angemessenes Vorgehensmodell definiert. In Bezug auf die Forschungsfragen muss dieses eine formale Basis bieten, um interdisziplinäre Entwicklungsaktivitäten zu verweben. Dabei bezieht sich die Formalisierung nicht nur auf eine formale Modellierung des Vorgehensmodells, sondern auch auf die Findung einer geeigneten Formalisierung von Domänenwissen und Anforderungen im Rahmen der Instanziierung eines solchen Vorgehensmodells.

Das Metamodell des Vorgehensmodell wird beschrieben (siehe Abschnitt 5.2.1), sowie ein geeigneter Weg zu dessen Instanziierung (siehe Abschnitt 5.2.2). Im Rahmen des Vorgehensmodells spielen OWL Ontologien eine zentrale Rolle, deren Zweck und Strukturierung in Abschnitt 5.3 beschrieben werden. Weiterhin wird das Vorgehensmodell bezüglich der gestellten Anforderungen verifiziert (siehe Abschnitt 5.4).

5.2 Vorschlag für die Prozessmodellierung

In Abschnitt 5.2.1 wird ein Vorgehensmodell präsentiert, in Abschnitt 5.2.2 dessen Instanziierung und technische Umsetzung.

5.2.1 Vorgehensmodell

Die Spezifikation domänenspezifischer Modelle im Rahmen des Domain und Requirements Engineering ist zentraler Part der Entwicklung innerhalb eines Entwicklungsstrangs. Weiterhin dienen diese Domänenmodelle und Produktmodelle der Verknüpfung heterogener Stränge. Basierend auf der Domänenontologie werden Anforderungen an die Entwickler in verschiedenen Strängen für den Entwurf gestellt. Diese Ontologien werden in der OWL formuliert und im Rahmen des Entwicklungsprozesses immer weiter formalisiert. Aufgrund der formalen Basis der OWL kann nun ein *Reasoner* verwendet werden, der automatisiert Schlussfolgerungen zieht (Inferenz). Dieses wird zur Konsistenzprüfung und zum Wissenstransfer verwendet. Dieser Ansatz ist in Abbildung 5-1 grafisch dargestellt.

Formale Basis des Vorgehensmodells

Domänenwissen umfasst nicht nur produktnahes Wissens, sondern umfasst ebenso Wissen bezüglich des Einsatzes von Methoden und Prozessen. Um die eben beschriebene Verwebung zu ermöglichen, ist die Formalisierung von Produkt- und Prozessmodellen notwendig. Im Weiteren wird hier die in den Grundlagen beschriebene OWL 2 verwendet, welche auf einer Beschreibungslogik fußt. Dabei wird sie genutzt, um Domänenwissen bezüglich eines interdisziplinären Entwicklungsprozesses auszudrücken, sowie die Beziehungen zwischen den verschiedenen Entwicklungssträngen. Dabei kann die Entscheidbarkeit der OWL dazu verwendet werden, das beschriebene Wissen auf Konsistenz zu prüfen. Des Weiteren können Schlussfolgerungen gezogen werden.

Somit wird auf Basis der OWL das Metamodell (abstrakte *Syntax*) des Vorgehensmodells beschrieben. Dazu werden Konzepte (Klassen) C definiert, dabei befindet sich der Name des

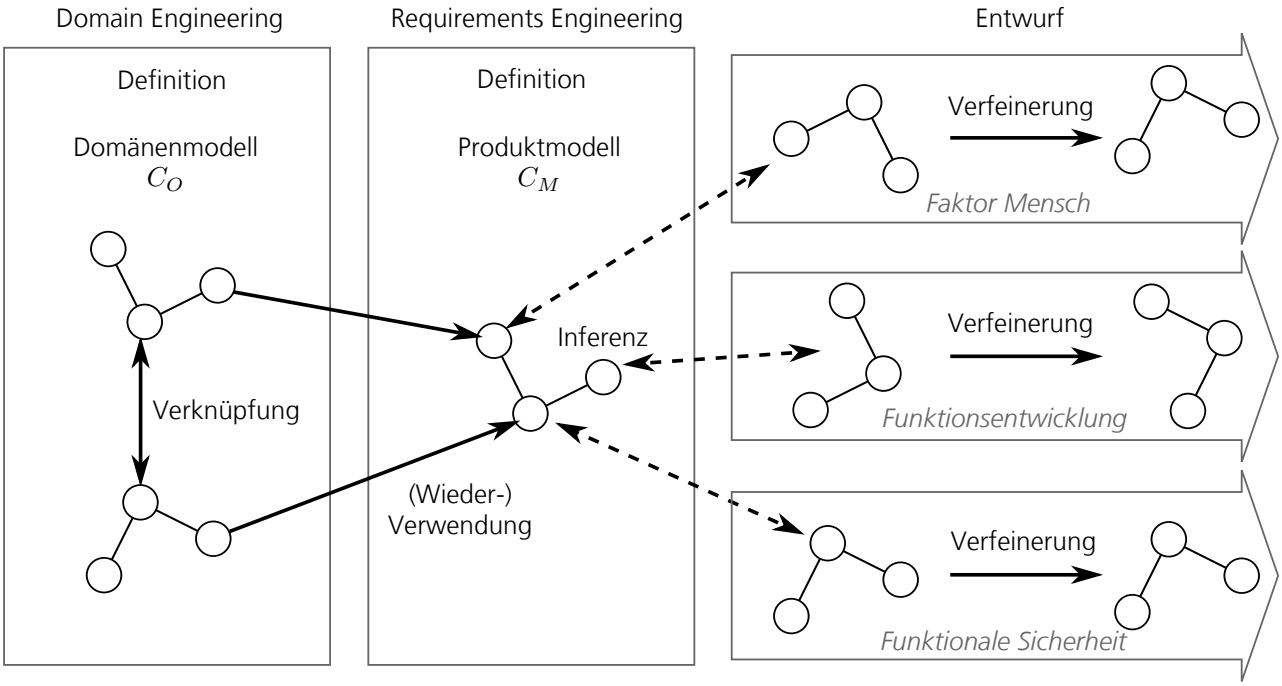

Abbildung 5-1: Die Spezifikation domänenspezifischer Modelle im Rahmen des Domain und Requirements Engineering auf Basis von formalisiertem Domänenwissen wird für die Verwebung der Entwicklungsstränge genutzt.

Konzeptes im Index (bspw. C_{Mensch}). Ähnlich werden Rollen (Relationen oder Eigenschaften) R geführt (bspw. $R_{\text{istKindVon}}$), oder wie Instanzen (Individuen) c (bspw. c_{Peter}). Darauf aufbauend wird die temporale *Semantik* des Vorgehensmodells unter Verwendung von Kripke Strukturen und LTL Formeln formalisiert. Die verwendete Notation ist in Anhang B ausführlich dargestellt.

Metamodell / Abstrakte Syntax

In diesem Abschnitt werden wesentliche Konzepte innerhalb des Entwicklungsprozesses (z.B. Phase, Iteration) und dazugehörige ordnende Relationen (z.B. „ist Phase innerhalb einer Iteration") eingeführt. Dabei werden notwendige (mittels des Teilmengenoperators $\sqsubseteq$) bzw. notwendige und hinreichende Bedingungen (mittels des Äquivalenzoperators $\equiv$) für die Inferenz gegeben, mit deren Hilfe eine Instanz einem Konzept eindeutig zugeordnet werden kann. Eine detaillierte Beschreibung der Syntax und Semantik der Beschreibungslogik findet sich im Anhang (siehe Abschnitt B.3).

Entwicklungsprozess Ein Entwicklungsprozess C_{DP} umfasst die Instanz des initialen Entwicklungsstrangs $c_{DS,0}^{[\![0]\!]}$:

$$C_{DP} \sqsubseteq R_{\text{comprises}}.\{c_{DS,0}^{[\![0]\!]}\} \text{ mit } C_{DS}(c_{DS,0}^{[\![0]\!]}) \tag{5.1}$$

sowie der initialen Iteration $c_{IT,0}^{[\![0]\!]}$:

$$C_{DP} \sqsubseteq R_{\text{comprises}}.\{c_{IT,0}^{[\![0]\!]}\} \text{ mit } C_{IT}(c_{IT,0}^{[\![0]\!]}) \tag{5.2}$$

Im Weiteren besteht der Entwicklungsprozess aus zwei miteinander verknüpften Teilmodellen. Iterationen und Entwicklungsphasen bilden die logischen und temporalen Abhängigkeiten und Inhalte einzelner Entwicklungsaktivitäten ab. Entwicklungsstränge umfassen das Domänenwissen der einzelnen Disziplinen und sind mit den Entwicklungsaktivitäten verknüpft.

Entwicklungsstrang Ein Entwicklungsstrang C_{DS} stellt die Arbeit einer Domäne innerhalb eines Entwicklungsprozesses dar. Neben Iterationen

$$C_{DS} \sqsubseteq R_{\text{worksOn}}.C_{IT} \tag{5.3}$$

umfasst ein Entwicklungsstrang seine Ontologie C_O,

$$C_{DS} \sqsubseteq R_{\text{comprises}}.C_O \tag{5.4}$$

die das Domänenwissen kapselt. Dazu kommen produktspezifische Modelle C_M,

$$C_{DS} \sqsubseteq R_{\text{comprises}}.C_M \tag{5.5}$$

auf dem Domänenwissen (bzw. der Ontologie C_O) aufsetzen. Weiterhin kann ein Entwicklungsstrang irreflexiv weitere Entwicklungsstränge umfassen.

$$C_{DS} \sqsubseteq R_{\text{comprises}}.C_{DS} \tag{5.6}$$

Eine Instanz $c_{\text{DS},j}^{[\![i]\!]}$ befindet sich auf der i-ten Granularitätsebene, und ist in dieser der j-te Strang.

Domänenontologie Eine Domänenontologie C_{DO} repräsentiert das Domänenwissen eines Entwicklungsstrangs C_{DS}, ist dabei eng mit der korrespondierenden Instanz $c_{\text{DS},j}^{[\![i]\!]}$ verknüpft. Die Ontologie wird im Laufe des Prozesses weiter formalisiert und liegt in einer formalen Form in OWL 2 vor. Das Domänenwissen erklärt Entwurfsartefakte eines Entwicklungsstrangs, bspw. die Bedeutung (Semantik) von Anforderungen im Kontext eines Entwicklungsstrangs.

Weiterhin werden Konzepte aus den zielorientierten Metamodellen aufgegriffen, so dass eine Verwebung auf der Ebene des Produktes weiter vereinfacht werden kann.

Ziel Ein Ziel C_{Goal} stellt eine abstrakte Größe dar, anhand dieser die Bedürfnisse von Kunden und anderen Quellen dokumentiert werden können. Ziele sind (noch) keine Anforderungen. Ziele können andere Ziele verfeinern.

$$C_{\text{Goal}} \sqsubseteq R_{\text{refines}}.C_{\text{Goal}} \tag{5.7}$$

Anforderung Ein Ziel kann zu einer Anforderung C_{Req} werden, nämlich genau dann, wenn ein Ziel einem Dienst C_{Service} zugeordnet wird.

$$C_{\text{Req}} \sqsubseteq R_{\text{satisfiedBy}}.C_{\text{Service}} \text{ mit } C_{\text{Req}} \sqsubseteq C_{\text{Goal}} \tag{5.8}$$

Dienst Ein Dienst C_{Service} umfasst Operationen, die ausgeführt werden, die an einen Dienst gestellten Anforderungen zu erfüllen. Bei einem Dienst C_{Service} kann es sich sowohl um einen technischen Dienst (Software und Hardware) handeln, genauso wie um eine Dienstleistung eines Entwicklers im Rahmen des Entwicklungsprozesses. Das genaue Verhalten wird über OWL-S spezifiziert werden.

$$C_{\text{Service}} \sqsubseteq C_{\text{ServiceOWLS}} \tag{5.9}$$

Methode Eine Methode C_{Method} kann zur Präzisierung und Erreichung eines Ziels verwendet werden.

Zu den Konzepten, die durch die Entwicklungsstränge gekapselt werden, kommen die Entwicklungsaktivitäten, welche die Entwicklungsstränge ausführen.

Iteration Eine Iteration C_{IT} umfasst einen oder mehrere Schritte zur Entwicklung innerhalb eines Strangs C_{DS}. Eine Iteration setzt sich aus Entwicklungsphasen zusammen:

$$C_{\text{IT}} \sqsubseteq R_{\text{comprises}}.C_{\text{DevPhase}} \tag{5.10}$$

Dabei können weitere Iterationen stattfinden.

$$C_{\text{IT}} \sqsubseteq R_{\text{comprises}}.C_{\text{IT}} \tag{5.11}$$

Der Fokus einer Iteration verschiebt sich, je näher der Start der Produktion rückt.

Entwicklungsphase Eine Entwicklungsphase C_{DevPhase} ist ein abstrakter Schritt endlicher Dauer innerhalb eines Entwicklungsprozesses C_{DP}. Dabei gibt es verschiedene, feste Typen von Entwicklungsphasen:

$$C_{\mathrm{DevPhase}} \sqsubseteq C_{DE} \sqcup C_{RE} \sqcup C_{DD} \sqcup C_{TV} \sqcup C_{PV} \sqcup C_{OR} \tag{5.12}$$

Innerhalb von Entwicklungsphasen können weitere Iterationen oder Entwicklungsphasen stattfinden:

$$C_{\mathrm{DevPhase}} \sqsubseteq R_{\mathrm{comprises}}.(C_{\mathrm{IT}} \sqcup C_{\mathrm{DevPhase}}) \tag{5.13}$$

Dabei werden verschiedene Entwicklungsphasen C_{DevPhase}, die im Folgenden erläutert werden, zu einer Iteration C_{IT} zusammengesetzt.

Die wesentlichen Entwicklungsphasen sind an das in [Bjø08, Bjø10] vorgeschlagene Modell für Softwareentwicklung angelehnt.

Domänenmodellierung Die Phase der Domänenmodellierung C_{DE} umfasst die explizite sukzessive Beschreibung von Domänenwissen. Handelt es sich um eine sicherheitskritische Funktion, findet hier vor allem explizit auch eine Formalisierung des Domänenwissens in der jeweiligen Ontologie $c^{[\![i]\!]}_{\mathrm{O},j}$ statt. Dieses umfasst eine produktbezogene, aber produktunabhängige, Modellierung der Anwendungsdomäne, ihrer Regeln, Konzepte und Relationen.

Anforderungsdefinition Basierend auf sukzessiv formalisiertem Domänenwissen werden in der Phase der Anforderungsdefinition C_{RE} Anforderungen definiert, die im jeweiligen Entwicklungsstrang C_{DS} und von seinen Teilentwicklungssträngen verstanden werden können. Hier wird produktunabhängiges Domänenwissen um produktspezifische Anteile ergänzt. Für nachfolgende Entwürfe kann solches Wissen zu Domänenwissen werden. Dieses ist dann der Fall, wenn das Produkt zu einem integralen Teil der Domäne wird, also etwa dann, wenn ein ACC System in nahezu allen Fahrzeugen verfügbar ist.

Entwurf Im Entwurf bzw. detaillierten Design C_{DD} geht ein Entwicklungsstrang seiner Kernarbeit nach, indem er die Spezifikation umsetzt und das Produkt seiner Arbeit gegen die Spezifikation verifiziert. Dabei können domänenspezifische Verfeinerungen in Richtung der Teilentwicklungsstränge erfolgen. Somit entsteht ein neuer semantischer Kontext (ausgedrückt durch den inkrementierten Index $i+1$). Dabei können innerhalb von $c^{[\![i]\!]}_{\mathrm{DD},j}$ durch die Entwicklungsstränge $c^{[\![i+1]\!]}_{\mathrm{DS},j,0} \cdots c^{[\![i+1]\!]}_{\mathrm{DS},j,\nu}$ mit $\nu \in N$ weitere Iterationen durchgeführt werden.

Test und Verifikation Das umgesetzte Produkt wird in der Testphase C_{TV} einer Verifikation unterworfen und muss die gestellten Anforderungen erfüllen, um in die nächste Phase zu gelangen.

Produktvalidierung Um die Sinnhaftigkeit von Anforderungen zu überprüfen, findet eine Validierung C_{PV} des Produktes statt. Hier wird überprüft, ob die Anforderungen, und damit der entstandene Prototyp, den Erwartungen gerecht werden. Dieses Wissen sollte für die folgende Iteration bei der Definition neuer Anforderungen genutzt werden.

Ontologieüberprüfung Der letzte elementare Schritt einer Iteration beschäftigt sich mit der Ontologieüberprüfung C_{OR}, einer kritischen Betrachtung der Modellierung des jeweiligen Domänenwissens. Dabei steht im Vordergrund, ob die beschriebenen Konzepte ausreichend und angemessen waren und es die Konzepte erlaubt haben, Anforderungen sinnvoll auszudrücken, so dass der Wunsch des Kunden adäquat in Anforderungen ausgedrückt werden konnte. Auch diese Betrachtung sollte in einer weiteren Iteration konstruktiv genutzt werden.

In Abbildung 5-2 ist ein Beispiel mit Bezug zur ISO 26262 dargestellt, übertragen auf die beschriebenen Konzepte. Der *Entwicklungsstrang* Funktionsentwicklung ist Teil des *Entwicklungsprozesses* für ein ACC, der gerade Iteration n durchläuft. Hier ist noch unklar, welche

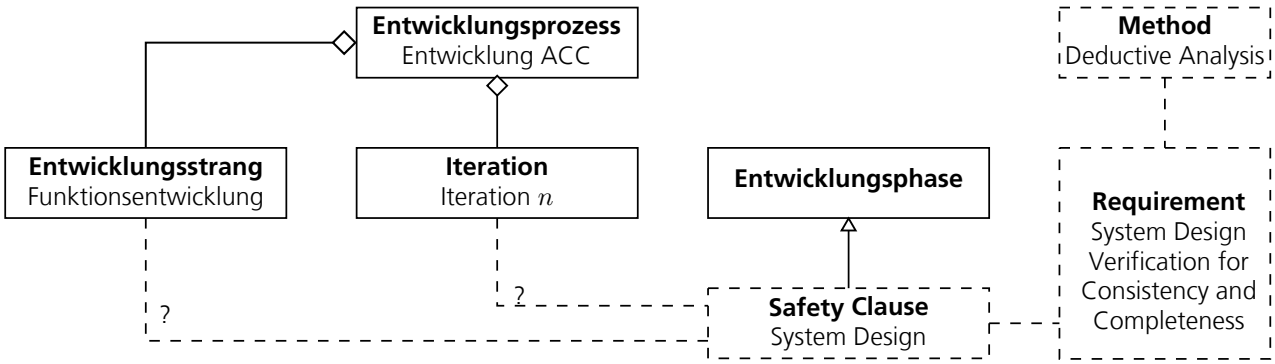

Abbildung 5-2: Vereinfachtes Beispiel aus der ISO 26262, übertragen in die Prozessontologie.

Teile der ISO 26262 Anwendung finden müssen, bspw. die Umsetzung der *Entwicklungsphase* System Design. Dieses wird durch die Durchführung einer automatisierten Risikoanalyse geklärt (siehe dazu auch das ausführliche Beispiel in Kapitel 8). Was jedoch in diesem Minimalbeispiel gut zu erkennen ist, wie die beschriebenen abstrakten Konzepte bzgl. eines Entwicklungsprozesses aufgegriffen werden können, um ein Vorgehensmodell mit spezifischen Begriffen in ein darüber liegendes Rahmenwerk einzubetten (z.B. *Safety Clause* als Spezialisierung einer *Entwicklungsphase*).

Prozessablauf / Temporale Semantik

Der Prozessablauf erfolgt auf Basis der beschriebenen Konzepte *iterativ* und *rekursiv* und beginnt im initialen Entwicklungsstrang $c^{[\![0]\!]}_{\mathrm{DS},0}(k)$, wobei es sich um die k-te Iteration handelt. Als erstes wird im *Domain Engineering* $c^{[\![0]\!]}_{\mathrm{DE},0}(k)$ die Domäne über die Ontologie beschrieben. Anschließend werden im *Requirements Engineering* $c^{[\![0]\!]}_{\mathrm{RE},0}(k)$ wichtige Eigenschaften der zu entwickelnden Produktes durch Ziele ermittelt und Anschließend beginnt der *Entwurf* in $c^{[\![0]\!]}_{\mathrm{DD},0}(k)$, und die Unterentwicklungsstränge beginnen ihre Iterationen. Danach beginnt der aufsteigende Teil des V-Modells, dabei werden die Ergebnisse der vorhergegangenen Arbeitsschritte untersucht und bewertet. Dieses beginnt mit dem Testen gegenüber der Anforderungsspezifikation in der Phase *Test und Verifikation* $c^{[\![0]\!]}_{\mathrm{TV},0}(k)$. Im Weiteren wird die Sinnhaftigkeit der Anforderungen während der Phase *Produktvalidierung* $c^{[\![0]\!]}_{\mathrm{PV},0}(k)$ überprüft. Abschließend für eine Iteration wird in der *Ontologieüberprüfung* $c^{[\![0]\!]}_{\mathrm{KR},0}(k)$ bewertet, inwiefern das Modell des Domänenwissens ausreichend und passend war. Die Ergebnisse dieser Analysen fließen dann in den Entwurf in der folgenden Iteration $c_{\mathrm{IT}}(k+1)$ ein. Dieser Ablauf ist auch in Abbildung 5-3 dargestellt. Neben dem generischen rekursiven Ablauf ist angedeutet, wie im Rahmen des Entwicklungsstrangs *Funktionsentwicklung* im Entwurf die Definition und Entwurf von System und Software enthalten sind. Weiterhin wird anhand des Zustands eines ACCs und dem Parameter der Fahrzeuggeschwindigkeit verdeutlicht, wie sich die Sicht auf diese Konzepte im Rahmen der Entwicklung ändert. Dabei kann ein Reasoner bei der Transformation unterstützen. Während es sich im Requirements Engineering noch um kontinuierliche bzw. diskrete Zustandsvariablen handelt, so werden diese in Software in entsprechenden Äquivalente (bspw. Fließkommavariablen) abgebildet.

Die Modellierung der Konzepte in OWL ermöglicht zwar die Prüfung der Konsistenz eines konkreten Vorgehensmodells in Bezug auf das konzeptuelle und abstrakte Prozessmodell, jedoch fehlt den modellierten Relationen eine temporale Bedeutung. Die Modellierung solch temporaler Relationen ermöglicht Aussagen bzgl. der Erreichbarkeit und insbesondere der Ausführbarkeit bzw. Instanziierbarkeit des konkreten Vorgehensmodells.

Dazu wird eine Modellierung der bereits beschriebenen Konzepte des Vorgehensmodells als Kripke Strukturen (siehe dazu auch Anhang B.1) mit totaler Transitionsrelation beschrieben. Diese

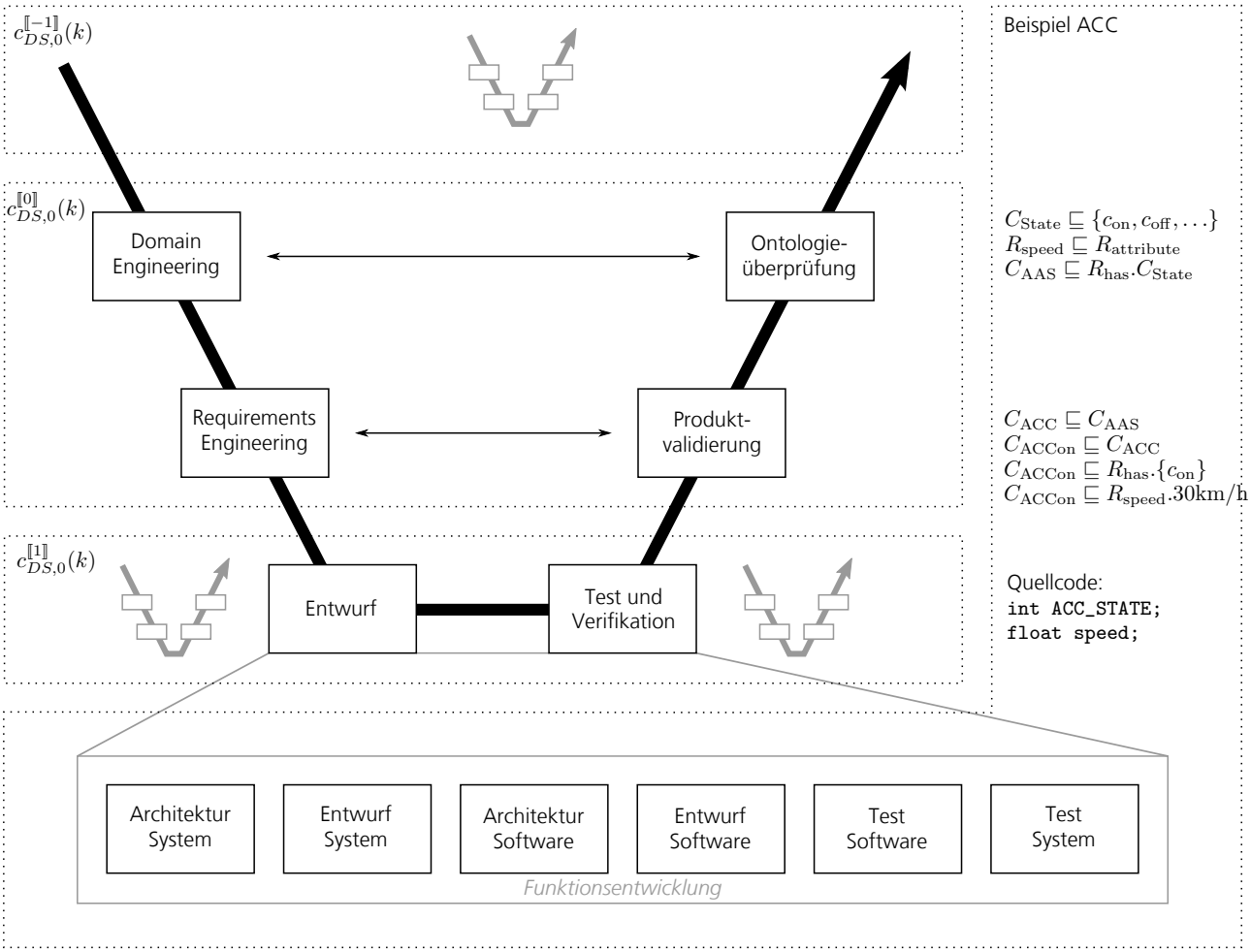

Abbildung 5-3: Das V-Modell, inklusive der Domänenmodellierung, welches jeder Entwicklungsstrang rekursiv durchläuft. Frühe Verifizierung verknüpft beide Seiten des V-Modells.

eignen sich insbesondere, da mächtige *Modelchecker* zur Verfügung stehen, die gewünschte Eigenschaften auf dem Modell nachweisen können. Diese Eigenschaften, wie etwa Erreichbarkeit, können etwa in LTL (Lineare Temporale Logik, siehe auch Anhang B.2) spezifiziert werden.

Prozess: Kripke Struktur Die Menge aller N konkreten Entwicklungsphasen stellt die Menge an Zuständen S der Kripke Struktur $K = (S, R, L, I)$.

$$C_{\mathrm{DevPhase}} \mapsto S \quad \text{bzw.} \quad c_{\mathrm{DevPhase},i} \mapsto s_i \in S, i \in \{1 \dots N\} \tag{5.14}$$

Durch die Zuweisung der mathematischen Aussagen $p_i \in AP$ wird jeder Zustand $s_i \in S$ über die Beschriftungsfunktion $L : S \to P(AP)$ eindeutig identifiziert.

$$\{p_i\} \subseteq L(s_i) \quad \text{mit} \quad c_{\mathrm{DevPhase},i} \mapsto p_i \in AP, i \in \{1 \dots N\} \tag{5.15}$$

Darüber hinaus ist die Markierung p_{ende} eines oder mehrerer finaler Zustände wichtig. Dabei dürfen solche Phasen $C_{\mathrm{FinalPhase}}$ über keinen Nachfolger verfügen.

$$c_{\mathrm{FinalPhase},i} \mapsto L(s_i) : \{p_{\mathrm{ende}}\} \quad \text{mit} \quad C_{\mathrm{FinalPhase}} \sqsubseteq= 0\ R_{\mathrm{next}}\ .\ C_{\mathrm{DevPhase}} \tag{5.16}$$

Weiterhin sind die Anfangszustände I zu benennen, diese Phasen $C_{\mathrm{FirstPhase}}$ dürfen keinen Vorgänger nachweisen.

$$C_{\mathrm{FirstPhase}} \mapsto I \quad \text{mit} \quad C_{\mathrm{FirstPhase}} \sqsubseteq= 0\ R_{\mathrm{last}}\ .\ C_{\mathrm{DevPhase}} \tag{5.17}$$

Die Zustandsübergänge R ergeben sich aus den Übergangsrelationen zwischen den einzelnen Phasen, wie in der Ontologie beschrieben.

$$R_{\mathrm{next}} \mapsto R \tag{5.18}$$

Dazu kommt die Standardrelation (s_i, s_i), welche jeden Zustand auf sich selber abbildet und somit die totale Transitionsrelation garantiert.

Hinzu kommt die Spezifikation gewünschter Eigenschaften. Diese werden in LTL formuliert (siehe dazu auch Anhang B.2). Anhand dieser Spezifikation können die zu definierenden Eigenschaften auf der eben beschriebenen Kripke Struktur bewiesen werden.

Prozess: LTL Spezifikation Die *Endlichkeit*, also die Tatsache, dass der Prozess unabhängig vom beschrittenen Pfad, global (G) und final (F) immer im Endzustand endet, lässt sich folgendermaßen fordern:

$$F(G(p_{\mathrm{ende}})) \tag{5.19}$$

Die *Erreichbarkeit* drückt aus, dass im Zuge des Prozessablaufs irgendwann alle Zustände des Prozesses s_i erreicht werden.

$$\forall i \in \{1 \dots N\} : F(p_i) \quad \text{mit} \quad \{p_i\} \subseteq L(s_i) \tag{5.20}$$

Ein Beispiel ist in Abbildung 5-4 anhand RESPONSE 3 CoP dargestellt. Wie in Abschnitt 2.2.1 beschrieben, fehlt hier die Verknüpfung der *Aktivität 5.6.3.A* zu ihrem logischen Vorgänger, der *Aktivität 5.6.2.C*. Durch diesen Fehler wird *Aktivität 5.6.3.A* nie erreicht (Erreichbarkeit) und nicht alle Pfade enden im Endzustand (Endlichkeit). Somit kann der genannte Fehler durch die Verifikation des Prozessmodells erkannt werden.

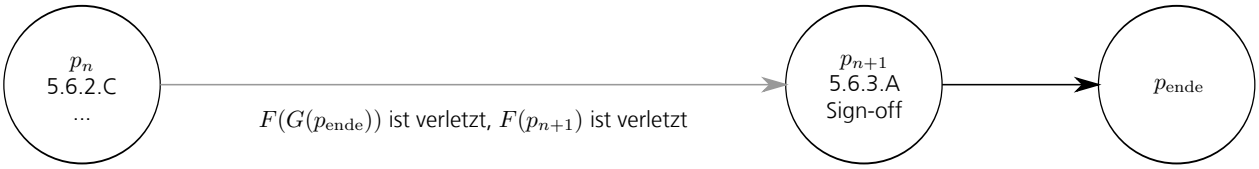

Abbildung 5-4: Beispiel für die Anwendung der Kripke Strukturen auf ein Beispiel aus RESPONSE 3 CoP.

5.2.2 Instanziierung und technische Umsetzung

Zur Ausführung und Instanziierung des Vorgehensmodells wird im Rahmen dieser Arbeit eine Abbildung eines konkreten Prozessmodells auf BPEL verwendet. Auf diese Weise kann ein konkretes Prozessmodell instanziiert werden. Für BPEL existieren Implementierungen, die eine Instanziierung unterstützen. Neben der Interpretation und Ausführung gehört dabei vor allem auch das Überwachung eines Prozesses und seiner Zustände, was wichtig für die Projektkontrolle ist.

Einzelne Entwurfsaktivitäten sind dabei als zustandslose Dienste modelliert, die im Rahmen des Prozesses eingebunden werden. Über die Eingabe und Ausgabe von Entwurfsartefakten wie Ontologien und Anforderungen nehmen sie auf den globalen Zustand des Prozesses Einfluss. Parallele Entwurfsaktivitäten können abgebildet werden, ebenso wie serielle Aktivitäten.

Technisch wird solch ein Prozessmodell aus einer Ontologie abgeleitet. Wie schon in [GJK+09] dokumentiert, bietet sich im Falle der XML basierten Serialisierung einer OWL Ontologie *eXtensible Stylesheet Language Transformations* (XSLT) als Technologie an, um aus dem OWL Modell konkrete Dokumentation zu generieren. Hinzu kommt die Erzeugung eines ausführbaren BPEL Workflows, wie im vorigen Kapitel beschrieben. Die Dokumentation basiert auf der Markupsprache HTML. Somit sind neben textuellen Elementen auch grafische Elemente eingebunden, die aus dem Prozessmodell abgeleitet sind. Dazu gehört die Dokumentation der Abhängigkeiten von Aktivitäten innerhalb des Entwurfsprozesses über Zustandsautomaten, die in den Normen beispielsweise noch über unübersichtliche textuelle Referenzen und eine tabellenartige Darstellung erfolgt. Darüber hinaus wird aus dem Prozessmodell ein Modell für NuSMV erzeugt. NuSMV ist ein Modelchecker zur automatischen Verifikation. Dieser überprüft das Prozessmodell (als Kripke Struktur) gegen die beschriebene LTL Spezifikationen.

Zu diesem Zweck wird in einem ersten Schritt aus dem OWL Modell eine proprietäre XML Darstellung erzeugt, die den Vorteil der einfacheren Nachverarbeitung bietet. Daraus können direkt Dateien für NuSMV bzw. BPEL erzeugt werden. Weiterhin wird aus dem XML basierten Prozessmodell eine HTML Darstellung generiert. Zudem werden auf *ant* basierende Automatisierungsartefakte erzeugt. Diese sorgen dafür, dass aus dem XML Prozessmodell über die Sprache *dot* gerichtete Grafen generiert werden. Diese visualisieren die Abläufe und können zu Bilddateien transformiert werden. Der technische Ablauf ist auch Abbildung 5-5 zu entnehmen.

5.3 Vorschlag für die Modellierung der Ontologien

Das Vorgehensmodell und die zugehörigen Ontologien sind zwar aus dem Kontext der Entwicklung sicherheitskritischer Assistenz- und Automationssysteme motiviert, jedoch sollen anwendungsunabhängige Teile generisch und wiederverwendbar gestalten werden. Eine Gesamtübersicht des Zuschnittes der Ontologien ist in Abbildung 5-6 zu sehen.

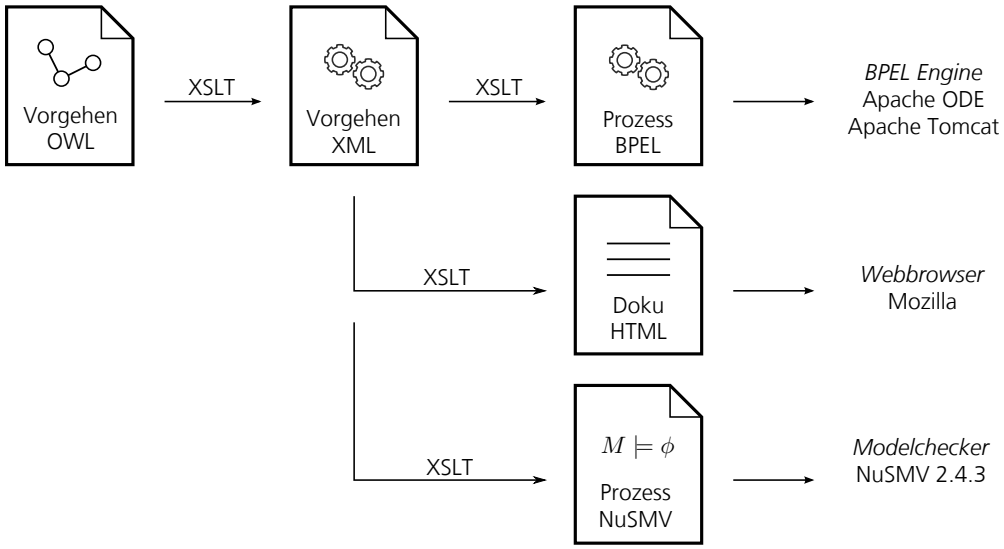

Abbildung 5-5: Technische Umsetzung eines Vorgehens auf ein ausführbares Format (BPEL), ein Verifikationsmodell (NuSMV) und Dokumentation (HTML).

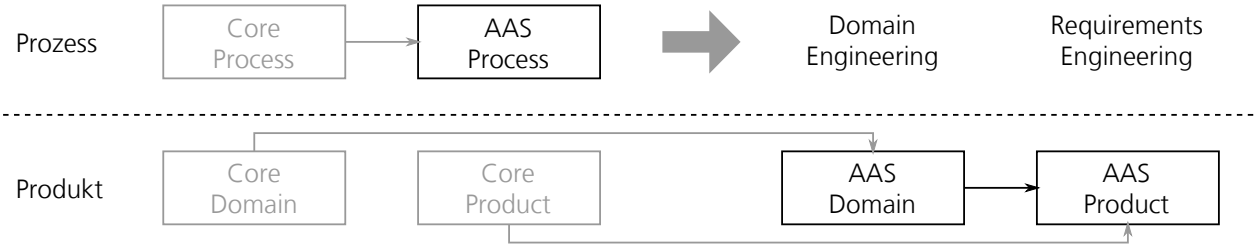

Abbildung 5-6: Im beschriebenen Vorgehensmodell spielen Domänenontologien eine wichtige Rolle, getrennt nach Prozess und Produkt.

5.3.1 Anwendungsunabhängige Ontologien

Anwendungsunabhängige Ontologien umfassen Konzepte, die generisch für Produktentwicklungen gelten. Weiterhin werden die Konzepte der Domäne, dem Produkt oder dem Prozess zugeordnet.

Core Process (Prozesskern) Diese Ontologie enthält alle Konzepte, die den Prozess betreffen. Diese sind auf einer anwendungsunabhängigen Granularität definiert. Elemente betreffen die Definition von Entwicklungssträngen, Phasen, etc., wie schon in Abschnitt 5.2.1 beschrieben.

Core Domain (Domänenkern) Diese Ontologie enthält alle Elemente, die die Domäne betreffen. Die in dieser Ontologie definierten Elemente beschreiben grundsätzliche physikalisch mathematische Zusammenhänge und eine Upper Ontology (vereinfacht aus SUMO), welche Konzepte, Attribute, Rollen etc. definiert. Basierend auf diesen Elementen können konkrete Domänen definiert werden.

Core Product (Produktkern) Diese Ontologie enthält alle Elemente, die das Produkt betreffen. Dieses sind domänenunabhängige Elemente. Vor allem wird hier das Produkt mit der Domäne verknüpft. Dabei werden Konzepte wie Ziele, etc. definiert.

5.3.2 Anwendungsabhängige Ontologien

Analog zu den anwendungsunabhängigen Ontologien werden AAS spezifische Verfeinerungen der Domäne, dem Produkt bzw. dem Prozess zugeordnet. Die genannten Verfeinerungen sind auch in Abbildung 5-6 dargestellt. Dabei schränken AAS Ontologien den Entwurfsfreiraum der Kernontologien weiter ein, formal gefasst durch Axiome in der OWL.

AAS Process (AAS Prozess) Diese Ontologie enthält Konzepte, die das Vorgehensmodell in der ADAS Domäne konkretisieren. Dazu gehört die Definition der bei der Entwicklung beteiligten drei zentralen Entwicklungsstränge.

AAS Domain (AAS Domäne) Diese Ontologie enthält Konzepte, welche die Domäne charakterisieren. Dazu gehören physische Objekte (Verkehrsteilnehmer, Straßen, Automobile und deren Komponenten), aber auch abstrakte Größen wie Verkehrsregeln und Gesetze. Die Konzepte ergeben sich insbesondere aus Domänenstandards, wie etwa der ISO 3833 [ISO77] oder auch die StVO [BMV07].

AAS Product (AAS Produkt) Diese Ontologie enthält Konzepte, die die Produktmodelle in der AAS Domäne konkretisieren. Dazu gehören z.B. Komponenten, die durch das neue System eingeführt werden. Das umfasst insbesondere Elemente im Fahrzeug (Hardware, Software). Dazu sind vor allem produktspezifische Standards herangezogen worden, wie etwa [ISO02b, ISO02a, ISO01a].

5.4 Verifikation des Vorgehensmodells

Im Weiteren wird beschrieben, wie die in diesem Kapitel vorgestellten Modelle auf die Anforderungen aus Abschnitt 4.4 wirken.

Das beschriebene Vorgehensmodell ist ein rekursives *V-Modell* (Anforderung V1). Dieses umfasst vor allem die Phasen des *Domain Engineering*, *Requirements Engineering*, sowie *Produktvalidierung* und *Ontologieüberprüfung*. Die Erstellung und Nutzung von Ontologien ermöglicht zusammen mit der Rekursion *Parallelisierung* (Anforderung V2) und *Synchronisation* (Anforderung V3) einzelner Entwicklungsaktivitäten. Darüber hinaus gehört ein rekursives Vorgehen zur Klasse der *iterativen* Vorgehensmodelle (Anforderung V4). Weiterhin können im Gegensatz zu anderen Vorgehensweisen mehrere Ebenen von *Granularität* (Anforderung V5) definiert werden.

Durch die *formale Basis* (Anforderung V8) in Form der abstrakten Syntax und temporalen Semantik kann die *Verifizierbarkeit* (Anforderung V7) umgesetzt werden. Dabei ist die abstrakte Syntax offen in OWL (Beschreibungslogik) definiert. Die temporale Semantik wird durch Kripke-Strukturen beschrieben, welche mittels LTL Formeln analysiert werden können. Dazu gehören Eigenschaften wie die Endlichkeit und Erreichbarkeit. Insbesondere durch die temporalen Eigenschaften und die Abbildung auf BPEL kann die *Ausführbarkeit* (Anforderung V6) gewährleistet werden. Weiterhin wurden die Prozessmodelle aus der ISO DIS 26262 und RESPONSE 3 modelliert (Anforderung V9) [GJB+09].

5.5 Zusammenfassung des Kapitels

Es wurde ein Vorgehensmodell vorgestellt, welches die Basis für *Formalisierung des Entwicklungsprozesses* stellt. Dazu werden Disziplinen in Entwicklungssträngen gekapselt. Wesentliches Element des Vorgehensmodells ist das *Domain Engineering*, welches unter Verwendung der OWL Domänenwissen formalisiert. Dieses wird im *Requirements Engineering* als Grundlage für die Spezifikation von Anforderungen verwendet. Dieses Vorgehen wirkt auf die *Verwebung interdisziplinärer Entwicklungsaktivitäten*, indem eine gemeinsame Anforderungsgrundlage entsteht. Dieses kann im Entwurf genutzt werden, der disziplinspezifische Aktivitäten umfasst. Entwicklungsstränge können weitere Teilentwicklungsstränge enthalten. Somit ist das Vorgehensmodell als *rekursives* Modell gestaltet. Weiterhin sind wie beim V-Modell Verifikationsschritte enthalten, die die Qualität sichern. Analog dazu ist eine Strukturierung von Ontologien vorgeschlagen, die Artefakte in verschiedenen Entwicklungsphasen aggregiert. So umfasst die *Core / AAS Domain*

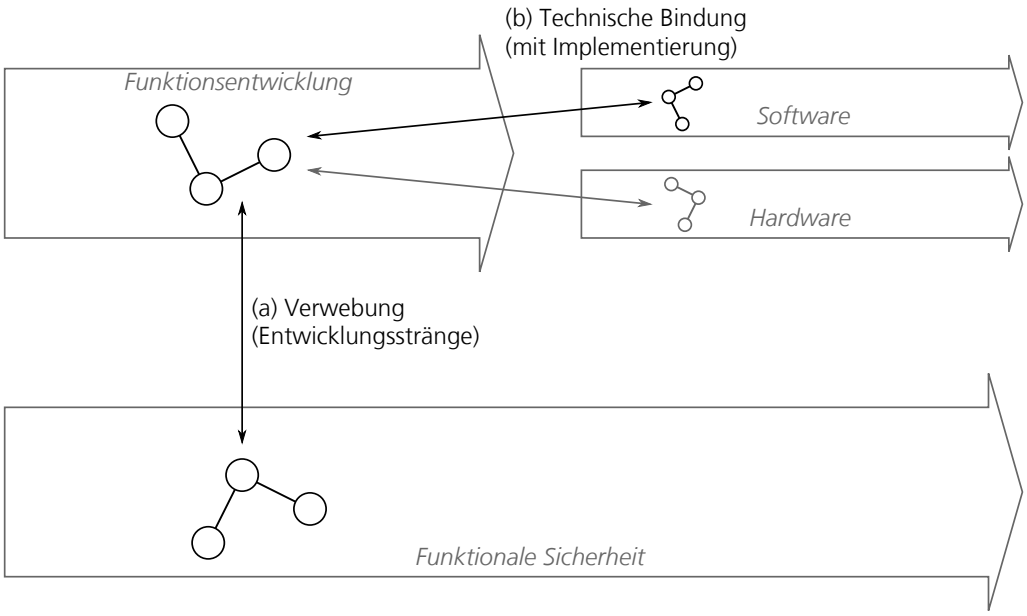

Abbildung 5-7: Aufbauend auf diesem Kapitel erfolgt die Verwebung zwischen Entwicklungssträngen (siehe Kapitel 6) und die technische Bindung (siehe Kapitel 7).

das im Domain Engineering gesammelte Wissen zur AAS Domäne. Die *Core / AAS Product* sammelt produktspezifisches Wissen, welches im Requirements Engineering gesammelt wird. Hinzu kommt mit *Core / AAS Process* prozessbezogenes Wissen. Wesentlich für die Verbindung zwischen Prozess- und Produktwissen im interdisziplinären Kontext ist weiterhin die Formulierung eines zielorientierten Ansatzes. Dabei kann ein Ziel zu einer Anforderung verfeinert werden, wenn sie an einen Dienst gebunden wird. Durch die Einbindung des Dienstes kann hier die OWL-S Ontologie aufgegriffen werden, die eine standardisierte Beschreibung von Diensten umfasst.

Die wesentlichen Konzepte des Vorgehensmodells wurden erläutert und in Beschreibungslogik formalisiert. Diese Formalisierung der abstrakten Syntax (Metamodell) ermöglicht die Konsistenzprüfung eines konkreten Modells, und kann weiterhin als Grundlage für eine Inferenz dienen. Diese Form der Formalisierung fasst die Begriffe eines Entwicklungsprozesses direkt. Zudem sind die wesentlichen Elemente als Kripke-Struktur beschrieben, welche somit den Konzepten eine temporale Semantik gibt. Dieses ermöglicht zusammen mit in LTL formulierten Randbedingungen die formale Prüfung eines konkreten Prozesses, insbesondere in Bezug auf Vollständigkeit und Terminierung. Eine Prozessinstanziierung erfolgt durch BPEL. Für eine prototypische Werkzeugunterstützung wurde dazu eine auf XSLT basierende Modelltransformation implementiert, die in OWL vorliegende Modellinstanzen des Vorgehensmodells auf BPEL bzw. NuSMV für die formale Analyse abbildet. Weiterhin kann Dokumentation in Form von HTML automatisch erzeugt werden.

Das beschriebene Vorgehensmodell liefert bei dessen Anwendung formalisiertes Domänenwissen, vor allem durch die Entwicklungsphase des Domain Engineering. Dieses Wissen wird im folgenden Kapitel 6 aufgegriffen, um Anforderungsmodelle zwischen verschiedenen Entwicklungssträngen miteinander zu verweben (siehe Abbildung 5-7). In Kapitel 7 wird auf die technische Bindung eingegangen. Das umfasst die Verknüpfung von Anforderungsmodellen mit Dienstemodellen, die eine Softwareimplementierung beinhalten. Dieses verdeutlicht, wie die abstrakten, konzeptuellen Modelle und Ontologien mit echtzeitfähigen Implementierungen zusammenspielen können. Dabei kommt ebenfalls formalisiertes Domänenwissen zum Einsatz.

6 Wissensbasierte Verwebung von Entwicklungsaktivitäten

6.1 Ziele des Kapitels

Nach der Einführung des formalen Vorgehensmodells und formalen Rahmenwerks wird in diesem Kapitel herausgearbeitet, wie das im Prozess gesammelte formale Domänenwissen aufgegriffen und für eine Reflexion von Entwurfsentscheidungen genutzt wird. Hierzu wird in Abschnitt 6.2 beschrieben, wie die Verwendung *formaler* Modelle einen Beitrag zur *Flexibilität* leisten kann.

Weiterhin wird in Abschnitt 6.3 die Verwebung von Anforderungsmodellen aus verschiedenen Domänen beschrieben. Im Sinne der Interdisziplinarität wird über diese Verknüpfung ein Wissenstransfer zwischen den Entwicklungssträngen ermöglicht. In Abschnitt 6.4 wird dieses genutzt, um Entwurfsentscheidungen innerhalb eines einzelnen Entwicklungsstrangs bezüglich ihrer Auswirkungen auf das gesamte Produkt und den Prozess zu bewerten. Dazu wird eine wissensbasierte Einflussanalyse modelliert. Zudem wird skizziert, wie das Ergebnis einer solchen Einflussanalyse verwendet werden kann, um ein Tailoring des Prozesses durchzuführen (Abschnitt 6.5).

6.2 Führung und Flexibilität durch Formalisierung

Durch das im vorangegangenen Kapitel beschriebene formale Vorgehensmodell sind zentrale Bestandteile eines Vorgehensmodells maschinenlesbar und mathematisch eindeutig beschrieben. Weiterhin sorgt das Vorgehensmodell dafür, dass im Rahmen der Systementwicklung Anforderungen sowie Domänenwissen immer weiter formalisiert und in Beschreibungslogik gefasst werden. Die Verwendung von OWL soll sicherstellen, dass die Modelle trotz Formalisierung *offen* bleiben.

Dieses ist zentrale Voraussetzung für eine erfolgreiche Verwebung von Domänenwissen. Denn über diese Offenheit muss ein Problembezug herstellbar sein, so dass generisches Domänenwissen für konkrete Projekte direkt nutzbar wird. Damit wird generisches Wissen einer Domäne für andere Entwickler verständlich gemacht, indem bestimmte Konzepte problembezogen aufgegriffen und verfeinert werden.

Die Verwendung eines wissensbasierten, formalen Ansatzes über die OWL zur Darstellung von Domänenwissen bietet nun die Möglichkeit, ausgehend vom Problem Schlussfolgerungen über anzuwendende Methoden und Maßnahmen zu treffen. Dieses wird dadurch erreicht, dass neben der Modellierung von Domäne und Produkt auch Prozesskonzepte modelliert werden (siehe Abschnitt 5.3).

Dabei leistet die *Formalisierung* einen Beitrag zur *Flexibilisierung*. Denn bei einer hohen Komplexität und großen Vielfalt anzuwendender Methoden und zu beachtender Anforderungen liefert die Formalisierung Beiträge zur Analysierbarkeit und Strukturierung, und somit Möglichkeiten zur Orientierung und Führung. Durch den wissensbasierten Charakter und die Modellierung des Problembezugs (Anwendungsdomäne) kann das Schlussfolgern einzelnen Entwicklern bei Entwurfsentscheidungen unterstützen. Denn durch den formalen Rahmen ist das partielle Wissen, über welches ein einzelner Entwickler verfügt, mit dem anderer Domänen verknüpft. Dadurch kann die Interaktion des einzelnen Entwicklers mit dem Wissen anderer Domänen helfen, Entwurfsentscheidungen gegeneinander abzuwägen, indem Konsequenzen in anderen

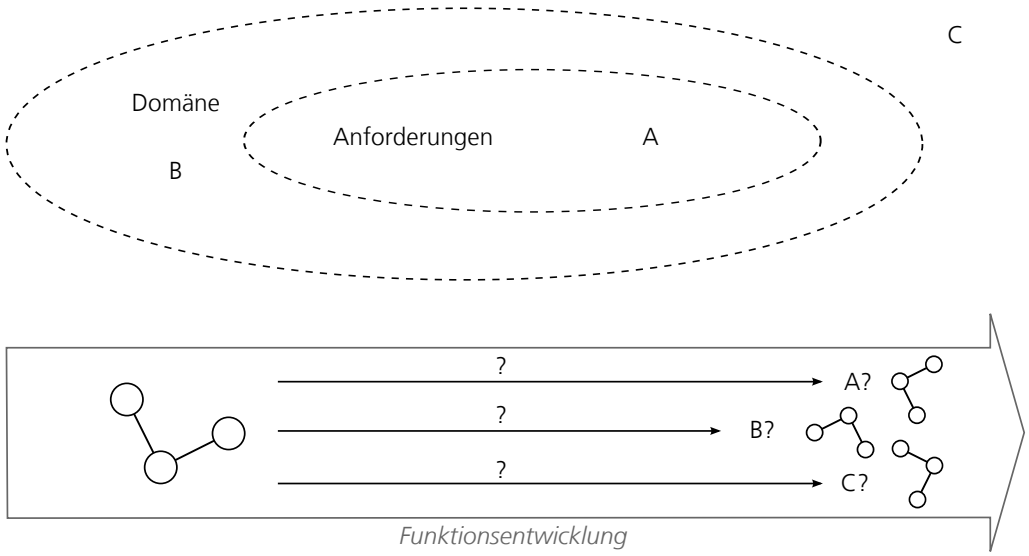

Abbildung 6-1: Formalisiertes Wissen bezüglich der Domäne und Anforderungen grenzen den Entwurfsfreiraum ein, und machen so Entscheidungen zwischen Entwurfsalternativen A, B und C einfacher.

Domänen aufgezeigt werden können. Somit kann durch die Formalisierung einzelnen Entwicklern die Struktur und Grenzen des Lösungsraums deutlich gemacht werden (siehe dazu auch Abbildung 6-1). Dieses kann konkret durch eine Einflussanalyse individueller Entwurfsentscheidungen auf das zu entwickelnde Produkt *und* den begleitenden Prozess umgesetzt werden. Letzteres wird im Rahmen eines allgemeinen Trends zur Prozessorientierung immer wichtiger, konkret vor allem bei Zulassungsprozessen sicherheitskritischer AAS.

Ein wichtiger wissensbasierter Aspekt solch einer Modellierung ist das Tailoring des Entwurfsprozesses, der sich aus spezifischen Entwurfsentscheidungen ergibt. Ändern Entwurfsentscheidungen etwa die Rahmenbedingungen einer sicherheitskritischen Funktion derart, dass ein anderer ASIL nach ISO DIS 26262 umgesetzt werden muss, so ändern sich neben Anforderungen für das Produkt auch Anforderungen an den Prozess bzw. geforderte Aktivitäten. Auch hier kann eine Inferenz auf den Ontologien zur Feststellung relevanter Produkt- und Prozessanforderungen verwendet werden.

6.3 Modellverwebung

Basis für eine Flexibilisierung durch Formalisierung ist eine sinnvolle Verwebung von Domänenmodellen und Anforderungsmodellen.

6.3.1 Ontologien und Metamodelle

Im konzeptionellen wie technischen Sinne stellt sich die Frage, wie die angesprochene Formalisierung, Verknüpfung und Inferenz zwischen Prozess, Domäne und Methoden erfolgen kann. Vor allem ist zu beantworten, wie die bereits angesprochenen *Ontologien* mit heute bereits existierenden modellbasierten Werkzeugen interagieren können, deren Modelle auf *Metamodellen* fußen.

Eine Ontologie ist eine explizite Spezifikation einer Konzeptualisierung [Gru95], und sie wird von mehreren Partnern geteilt. Metamodelle beschreiben explizit die Konstrukte (Syntax), Regeln (Einschränkungen) und Semantik, die benötigt wird, um Modelle innerhalb einer bestimmten Domäne zu definieren. Die Unterscheidung zwischen Metamodellen und Ontologien ist in der

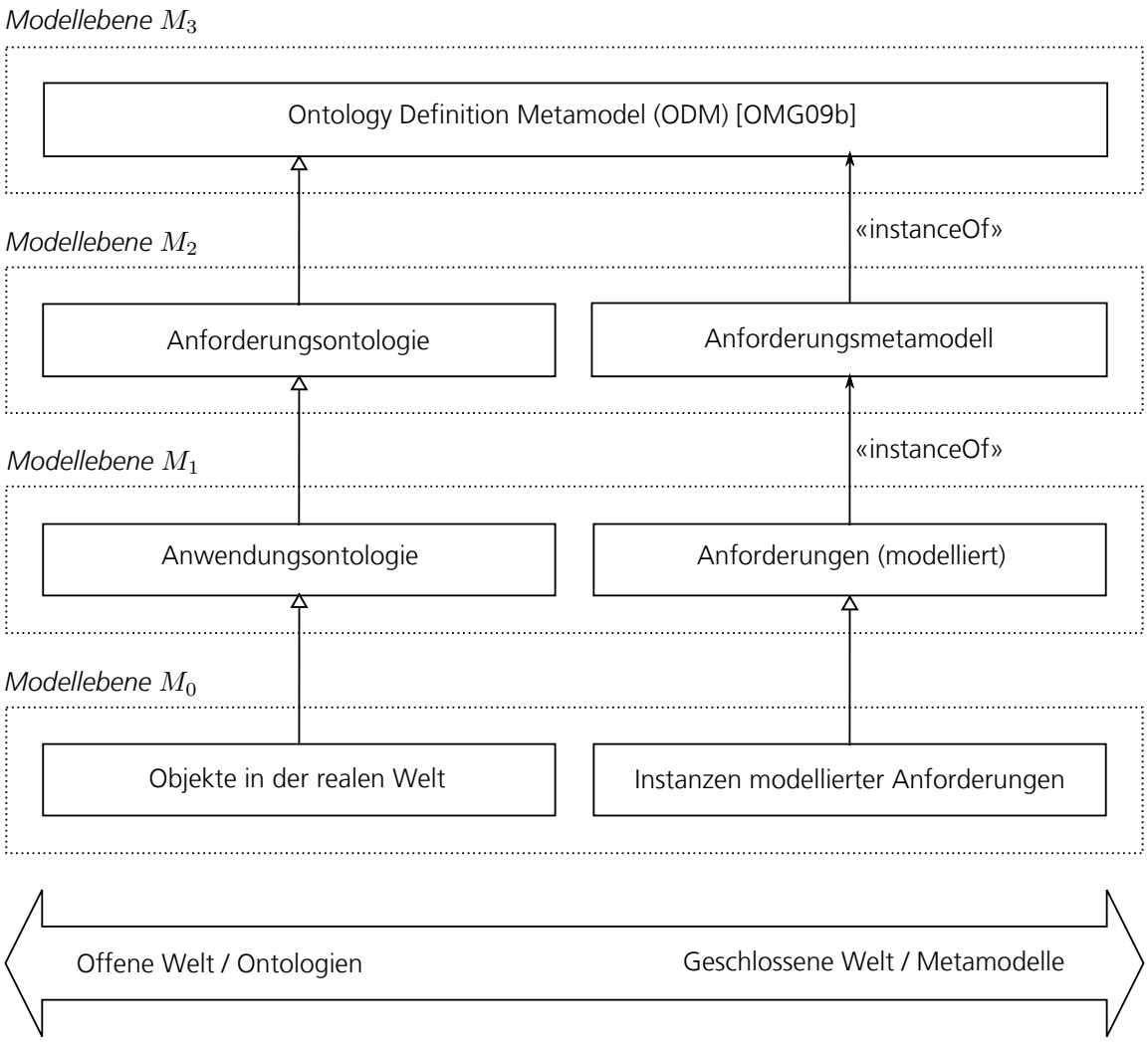

Abbildung 6-2: Die Ontologie hat ihren Schwerpunkt in der deskriptiven Analyse einer Domäne, das Metamodell in der präskriptiven (technischen) Spezifikation.

Literatur nicht einheitlich beschrieben, da beide Modellierungsansätze für ähnliche Tätigkeiten verwendet werden *könnten*. Die Unterschiede lassen sich einfach verstehen, wenn man den Hintergrund beider Technologien betrachtet: Ontologien im Kontext der Informationstechnik haben ihre Wurzeln insbesondere im Feld der Wissensrepräsentation künstlicher Intelligenz. Solch eine Repräsentation muss vor allem automatisiertes Schlussfolgern (Inferenz) ermöglichen. Dagegen kommen Metamodelle vor allem im Bereich des Softwareengineering zum Einsatz, insbesondere zur Modellbeschreibung im Bereich modellgetriebener Architekturen und Vorgehensweisen.

Im Weiteren kommt eine abgewandelte Form der Ideen von [AZW06] zum Einsatz. Diese schlagen den Einsatz von Ontologien als Integration von Metapyramiden vor. Dabei werden sowohl Ontologien als auch Metamodelle entsprechend ihrer jeweiligen Stärken genutzt. Der Kernaspekt einer Ontologie liegt in der *Analyse* einer Domäne, während das eigentliche Metamodell das Resultat einer Entwurfsaktivität darstellt. Es entsteht eine konkrete (technische) *Spezifikation* für Modelle der betroffenen Anwendungsdomäne. Ihrer Natur nach ist die Ontologie eher *deskriptiv* (beschreibend), weniger *präskriptiv* (vorschreibend), wie es das Metamodell ist. Anschaulich ist diese Rolle in Abbildung 6-2 dargestellt.

Solch explizite Trennung zwischen Analyse (und Beschreibung) der Requirements Domäne und dem Entwurf (und Spezifikation) des Metamodells hat verschiedene Vorteile. So ist etwa die Ontologie durch ihren beschreibenden Charakter keinen technischen Optimierungszwängen unterlegen (insbesondere vor dem Hintergrund einer Werkzeugentwicklung). Dieses erlaubt

eine saubere konzeptionelle Beschreibung. Dadurch besteht eine hohe Chance einer Wiederverwendung der Ontologie in einem anderen Projekt, auch dann, wenn technisch keine Möglichkeit zur direkten Wiederverwendung des Metamodells besteht. Weiterhin kann durch die Verwendung der OWL 2 (Web Ontology Language), die auf der Beschreibungslogik SROIQ basiert, die Domänenontologie unter Nutzung eines Reasoners analysiert werden. Diese kann etwa zur Aufdeckung von Redundanzen und Inkonsistenzen in einem Domänenmodell genutzt werden. Dieses wiederum unterstützt direkt die Definition des Metamodells.

Um die Integration von Ontologien in die Welt der modellgetriebenen Architekturen zu vereinfachen, hat die Object Management Group die Spezifikation des *Ontology Definition Metamodel* [OMG09b] veröffentlicht:

> „The ODM is applicable to knowledge representation, conceptual modeling, formal taxonomy development and ontology definition, and enables the use of a variety of enterprise models as starting points for ontology development through mappings to UML and MOF."

Ein besonderer Aspekt der OWL 2 ist die sogenannte *Open World Assumption*. Diese sagt aus, dass der logische Wert einer Aussage unabhängig davon ist, ob sie für einen einzelnen Beobachter als wahr bekannt ist. Dieses ist das Gegenteil der *Closed World Assumption* (CWA). Diese nimmt an, dass eine nicht explizit formulierte Aussage falsch ist. Die Nutzung der Open World Assumption ist insbesondere hilfreich in stark verteiltem und heterogenem Arbeitsumfeld, wenn offene und erweiterbare Domänenmodelle beschrieben werden.

Beispiel CESAR RMM

Ein Beispiel für solch ein offenes Metamodell ist das *Requirements Meta-Model* (RMM), welches im Kontext des europäischen Projektes CESAR (siehe auch Abschnitt 1.3.2) entwickelt wird und nur sehr wesentliche Konzepte des Requirements Engineering (z.B. *Requirement*, *Stakeholder*) fasst. Zusätzlich kommt eine Ontologie zum Einsatz, die die Domäne des Requirements Engineering beschreibt und die Konzepte des Metamodells weiter verfeinert. Die Ontologie enthält dabei unter anderem eine detaillierte Kategorisierung von Anforderungstypen (z.B. die Kategorie *Performance Requirement*, die wiederum bspw. ein *Timing Requirement* umfasst). Diese Kategorisierung und Nomenklatur kann in jedem Unternehmen verschieden sein. Somit lässt sich das Metamodell jedoch universell und flexibel einsetzen. Mehr Details hierzu können bei [ML10] nachgelesen werden.

6.3.2 Verweben von Requirements-Modellen

Wie das Beispiel CESAR verdeutlicht, existiert im interdisziplinären Bereich des Requirements Engineering eine Vielzahl von Methoden, und damit in ihrer formalisierten Form auch eine Vielzahl von Metamodellen. Daher ist eine vollständige Standardisierung nicht mehr sinnvoll, da zu viele Anwendungen eingeschränkt werden würden.

Die eben beschriebene parallele Verwendung von Metamodellen und Ontologien kann nun dazu verwendet werden, verschiedene Metamodelle auf der Ebene der zugehörigen Ontologien zu analysieren. Auf dieser Ebene können diese gegebenenfalls semantisch verknüpft werden. Der Vorteil ist, dass auf diese Art und Weise beide Domänenmodelle unabhängig voneinander verwendet und relevante Anteile in eine jeweils andere Domäne überführt werden können. Da der Abgleich auf der Ebene der Konzepte bzw. Klassen erfolgt, kann dieses Vorgehen generisch für unterschiedliche Modellinstanzen verwendet werden.

Beispiel GOMS und KAOS

Einfach nachvollziehen lässt sich die Idee an den Metamodellen von GOMS und KAOS, wie sie in Abschnitt 3.2 beschrieben sind. GOMS lässt sich insbesondere während der Gewinnung von

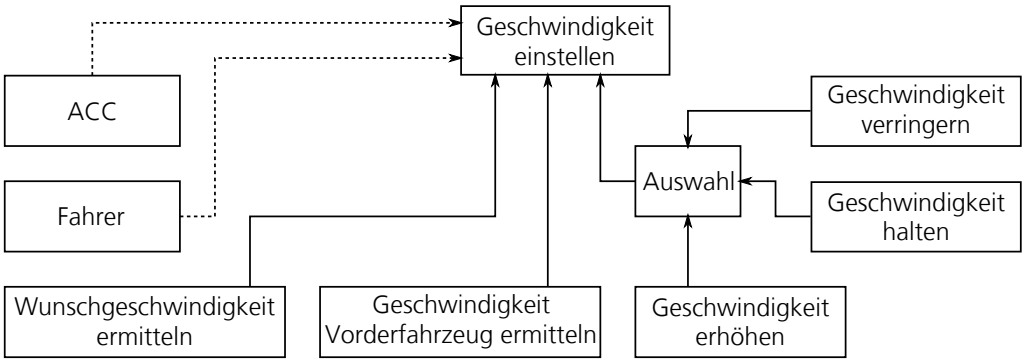

Abbildung 6-3: Beispiel für ein gemeinsames Zielmodell zwischen Fahrer und System (ACC).

Anforderungen einsetzen, KAOS hingegen während der Dokumentation bzw. Spezifikation von Anforderungen.

GOMS fasst insbesondere Aspekte, die sich aus der Bedienung eines technischen Systems ergeben. Für die Anwendung von GOMS muss noch kein technischer Prototyp vorliegen. Möchte man nun den Prototyp technisch genauer spezifizieren, so kann dieses durch die Beschreibung einer Zielstruktur erfolgen, der ein menschlicher Operator folgt. Diese Zielstruktur kann dann in KAOS wiederum aufgegriffen werden, wenn eine für einen technischen Entwickler verständliche Spezifikation entsteht. Weiterhin kann die so dokumentierte Zielstruktur und die Allokation auf einen menschlichen Operator bzw. ein technisches System genutzt werden, einzelne Aufgaben, und damit verbundene Ziele, vom Operator auf ein technisches System zu verlegen. So kann eine Automatisierung durchgeführt werden, ohne die Aufgabenstruktur an sich zu verändern. Dabei können die Ontologien alternativ zu direkten modelltransformierenden Verfahren (z.B. OMG QVT Standard für MOF Modelle) verwendet werden. Der Vorteil von OWL liegt hier in der offenen, formalen Basis. Neben der reinen OWL Logik kann auch etwa die *Semantic Web Rule Language* (SWRL) oder die *SPARQL Protocol and RDF Query Language* (SPARQL) verwendet werden, um Modelle ineinander zu überführen.

Aufgrund der hohen syntaktischen und semantischen Ähnlichkeit von GOMS und KAOS eignen sich diese Modelle besonders gut zum Einsatz in einem interdisziplinären Kontext. Ein Beispiel ist in Abbildung 6-3 zu sehen. Dabei ist das Einstellen einer bestimmten Geschwindigkeit eine Aufgabe, die vom Fahrer auf das Fahrzeug (ACC) übergeht. So kann ein GOMS Modell, welches das Verhalten des Fahrers abbildet, um Aspekte erweitert werden, die eine Nutzung in KAOS als Anforderungsspezifikation für das zu entwickelnde System ermöglichen. Insbesondere die hierarchische Zielstruktur kann wiederverwendet werden.

6.4 Einfluss von Entwurfsentscheidungen

6.4.1 Aufdeckung impliziter Abhängigkeiten

Neben einer Verknüpfung von Anforderungen mit anderen Anforderungen können Anforderungen auch mit Artefakten einer Anwendungsdomäne verknüpft sein. Dieser Bezug zur Anwendungsdomäne an sich kann dann wiederum indirekt Anforderungen umfassen. Ein Beispiel hierfür sind Anforderungen, die sich aus einer Norm oder einem Standard ergeben. Solch ein Bezug zur Anwendungsdomäne kann dazu genutzt werden, um per Inferenz Anforderungen für ein Produkt in der Entwicklung abzuleiten. Somit können aus implizit vorhandenen Eigenschaften wiederum explizite Anforderungen formuliert werden, die sich aus der Anwendungsdomäne ergeben.

Die Aufdeckung solcher impliziten Eigenschaften und daraus resultierender neuer Anforderungen ist elementarer Teil einer Einflussanalyse. Dabei stellt sich die Frage:

> „Was muss alles beachtet werden, wenn das Produkt über neue oder geänderte Eigenschaften verfügen soll? Was ergeben sich für Anforderungen an den Entwurf, die nicht direkt offensichtlich sind?"

Rechtzeitige Aufdeckung solcher impliziten Anforderungen ist vor allem interessant, um möglichst früh in der Entwicklung Entwurfsalternativen abzuwägen. Eine technisch möglicherweise einfache Änderung mag sich durch die komplexen Abhängigkeiten, insbesondere durch vorgeschriebene Sicherheitsmaßnahmen, später im Entwurf als eine weniger wünschenswerte Variante herausstellen.

Um solche impliziten Anforderungen in einem interdisziplinären Umfeld leichter entdecken zu können, kann die Verknüpfung der Domänen über die Ontologien genutzt werden. Ein verallgemeinertes Vorgehen kann wie in Abbildung 6-4 aussehen:

1. *Definition* von Eigenschaften des Produktes mit Bezug auf den eigenen Entwurfsstrang. Dabei charakterisiert der Entwickler sein zu entwickelndes System. So können konkrete technische Anforderungen beschrieben werden, etwa die Betriebsgrenzen eines ACC zwischen 30 km/h und 180 km/h.
2. *Annotation* der Eigenschaften des Produkts in Bezug auf die Anwendungsdomäne. Im Falle eines ACC könnte der Entwickler beschreiben, dass solch ein System auf der Landstraße und auf Autobahnen aktiv sein soll, nicht jedoch im Stadtverkehr. Im konkreten Fall könnte dieser Schritt auch schon per Inferenz gelöst sein, etwa weil sich aus der Minimalgeschwindigkeit ergibt, dass das ACC nicht im Stadtverkehr sinnvoll einsetzbar ist.
3. *Inferenz* der Konsequenz in anderen Entwicklungssträngen. Diese kann im Beispiel des ACC sein, dass ein Teil des Systems sicherheitskritisch wird und eine ASIL Klassifizierung erhält.
4. *Rückfluss* der Konsequenz für den eigenen Entwicklungsstrang. Dieses umfasst die Rückwirkungen auf den ursprünglichen Entwicklungsstrang, die durch den im vorhergehenden Schritt induzierten Einfluss im anderen Entwicklungsstrang zustande kommen. Zum Beispiel kann dies bedeuten, dass durch die ASIL Klassifikation weitere Maßnahmen bei der Entwicklung notwendig werden.

Der Vorteil einer formalen Modellbildung bedeutet in diesem Kontext, dass genau solche Analysen auf den Modellen automatisiert durchgeführt werden können.

6.4.2 Einflussanalyse mit der OWL

Um solch eine automatisierte Analyse auf Basis der OWL zu verdeutlichen, wird an dieser Stelle eine ASIL Einflussanalyse mittels OWL beschrieben. Da es sich um eine Analyse im Kontext der Funktionalen Sicherheit handelt, ist die formale Basis der OWL besonders geeignet. Im Speziellen profitiert die Modellierung in OWL von der Möglichkeit, auf Relationen Transitivität zu formulieren. Damit gilt (siehe auch Abschnitt B.3):

$$\langle c_1, c_2 \rangle \in R^{\mathcal{I}} \wedge \langle c_2, c_3 \rangle \in R^{\mathcal{I}} \rightarrow \langle c_1, c_3 \rangle \in R^{\mathcal{I}} \tag{6.1}$$

Bei der ASIL Einflussanalyse ist herauszufinden, ob sich durch die Vernetzung mit anderen Komponenten zusätzliche Sicherheitsanforderungen an die zu entwickelnde Komponente ergeben. Solch eine Analyse kann einfach aufgegriffen werden, wenn die Abhängigkeiten bereits implizit über die Domäne modelliert sind. Ein Beispiel hierfür ist etwa die Modellierung des

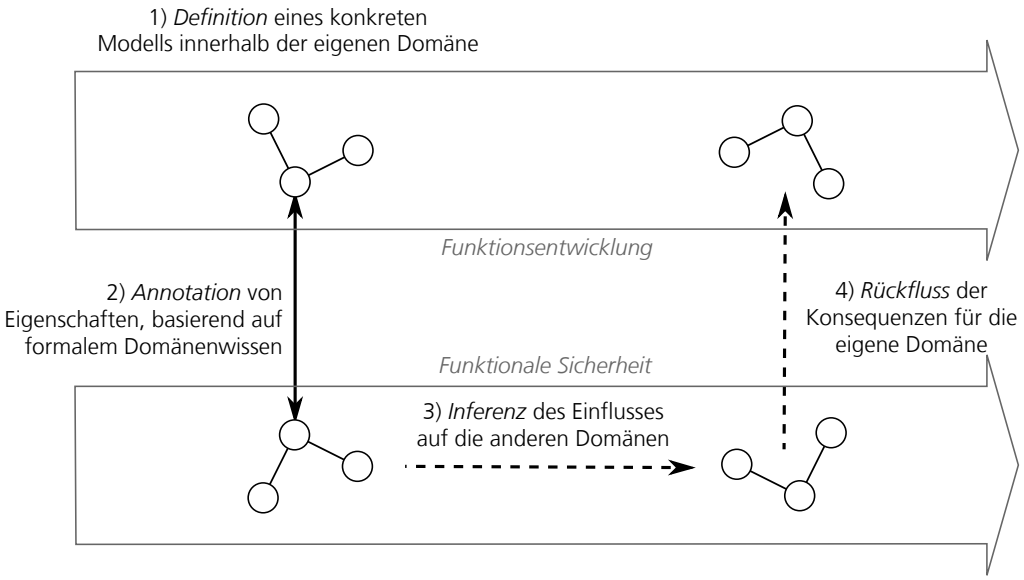

Abbildung 6-4: Aus den verknüpften Modellen verschiedener Domänen werden mit Hilfe eines Reasoners Schlussfolgerungen über Konsequenzen in der eigenen Domäne gezogen.

Informationsflusses zwischen Komponenten einer Systemarchitektur, der für eine Bestimmung von Abhängigkeiten herangezogen werden kann.

Es existieren vier ASIL Einstufungen $C_{\mathrm{ASIL}}(c_A)$, $C_{\mathrm{ASIL}}(c_B)$, $C_{\mathrm{ASIL}}(c_C)$ und $C_{\mathrm{ASIL}}(c_D)$. Daher kann die Klasse für den ASIL zu

$$C_{\mathrm{ASIL}} \equiv \{c_A, c_B, c_C, c_D\} \tag{6.2}$$

definiert werden. Einer einzelnen Komponente kann ein bestimmter ASIL zugewiesen werden, wenn sie Teil einer sicherheitskritischen Funktion ist. Gleichzeitig kann solch eine Komponente durch bereits getroffene Maßnahmen einen bestimmten ASIL zusichern. Nun können hier im Wesentlichen zwei Dinge untersucht werden:

- Welche ASILs bekommt eine Komponente durch den Systemzusammenhang indirekt zugewiesen?
- Erfüllt eine Komponente bereits den nötigen ASIL?

Dabei ist eine Komponente definiert durch einen ASIL, den sie zusichert.

$$C_{\mathrm{Component}} \sqsubseteq (= R_{\mathrm{assertsASIL}} \,.\, C_{\mathrm{ASIL}}) \tag{6.3}$$

Dazu kommt ein ASIL, den sie durch eine externe Vorgabe erfüllen muss.

$$C_{\mathrm{Component}} \sqsubseteq (= R_{\mathrm{hasToFulfillASIL}} \,.\, C_{\mathrm{ASIL}}) \tag{6.4}$$

Dadurch werden Aussagen, dass eine Komponente die direkt an sie gestellten Forderungen nicht erfüllt, vereinfacht:

$$C_{\mathrm{UnfinishedComponentB}} \sqsubseteq (= R_{\mathrm{asserts}} \,.\, c_A) \sqcup (= R_{\mathrm{hasToFulfillASIL}} \,.\, c_B) \tag{6.5}$$

Dieses bezieht sich bisher nur auf eine isolierte, einzelne Komponente mit einer einzigen externen Sicherheitseinstufung. Die einzelnen Komponenten können nun auch miteinander vernetzt sein, dabei ergeben sich Abhängigkeitsbeziehungen. Die Definition einer vernetzten Komponente ergibt sich folglich daraus, dass sie von mindestens einer anderen Komponente abhängt oder von mindestens einer anderen Komponente benötigt wird:

$$C_{\mathrm{SystemComponent}} \sqsubseteq (>= 1 R_{\mathrm{dependsOn}} \,.\, C_{\mathrm{Component}}) \sqcap (>= 1 R_{\mathrm{neededBy}} \,.\, C_{\mathrm{Component}}) \tag{6.6}$$

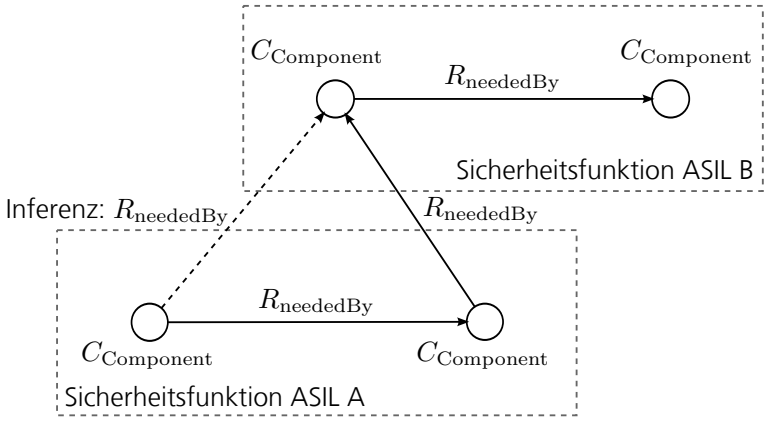

Abbildung 6-5: Abstrakte Illustration zur Modellierung von transitiven Abhängigkeiten, die mit Hilfe eines Reasoners und Beschreibungslogik modelliert werden können.

Weiterhin werden dadurch auch logisch die Anforderungen weitergegeben, so dass sich die Definition einer unterentwickelten Komponente ändert:

$$C_{\text{UnfinishedSystemComponentB}} \sqsubseteq (= R_{\text{asserts}} \,.\, c_A) \sqcup (>= 1 R_{\text{neededBy}} \,.\, (= R_{\text{hasToFulfillASIL}} \,.\, c_B)) \quad (6.7)$$

Dies bedeutet, dass neben der offensichtlichen aus sich selber abgeleiteten Eigenschaft berücksichtigt werden muss, ob ein anderes sicherheitskritisches System die aktuelle Komponente für ihre Funktionalität benötigt. Die transitive Eigenschaft der Relation R_{neededBy} garantiert, dass diese Relation entsprechend weitergegeben wird. Ein Reasoner kann nun dieses Modell prüfen und beweisen, ob es Komponenten gibt, die noch weiterentwickelt werden müssen.

Für das angesprochene Problem gibt es auch algorithmische Lösungen. Die hier vorliegende hat insbesondere den Vorteil, dass durch Wiederverwendung bereits existierender Verknüpfungen Aussagen für die Analyse getroffen werden können. Wenn bspw. eine Architektur beschrieben wird, liegen die Abhängigkeitsbeziehungen zwischen den Komponenten bereits implizit vor. Was formalisiert werden muss, ist, dass diese Abhängigkeitsbeziehungen um ihre Semantik im Kontext der Einflussanalyse erweitert werden. Dieses geschieht durch die Formulierung der eben beschriebenen Axiome in OWL. Diese Erweiterung wird durch die *Offenheit* der OWL möglich.

Weiterhin liegt in der logischen Modellierung ein mathematischer Beweis für die Klassifikation der Komponenten vor (etwa als unterentwickelte Komponente). Somit ist die erwähnte Analyse verglichen mit einer algorithmischen Lösung einfach, denn diese müsste rekursive Abhängigkeiten auswerten. Hinzu kommt, dass mit einem Reasoner wie HermiT und dem Editor Protégé generische Werkzeuge zur Verfügung stehen. Dabei verfügt Protégé insbesondere über die Eigenschaft, aus dem mathematischen Modell Restriktionen für die textbasierte oder grafische Bearbeitung eines solchen Modells abzuleiten. Diese können dann direkt bei der Eingabe geprüft werden. Ein Beispiel hierfür ist, dass eine Systemkomponente immer mit mindestens einer anderen Komponente verbunden sein muss. Denn sonst kann es sich nicht um eine Systemkomponente handeln.

Die generische Modellierung hat jedoch auch Kehrseiten. Dieses sind insbesondere zwei Aspekte. Unter dem Aspekt der *Komplexität* benötigen große Ontologien unter Umständen eine sehr hohe Rechenleistung für die Inferenz. Dieses kann ggf. auf aktuellen PCs nicht in akzeptabler Zeit ausgeführt werden. In einem solchen Fall können die Profile (EL, RL und QL) der OWL verwendet oder SWRL und SPARQL genutzt werden. So kann nach wie vor ein Beschreibungsmittel wie OWL und RDF verwendet werden, nur die Inferenz würde technisch effizienter implementiert. Bezüglich *Ausdruck* bei der Modellierung ist auf die Besonderheiten der *Open World Assumption* zu achten. Die Offenheit geht zulasten der Ausdrucksstärke, also dem was aus den formulierten

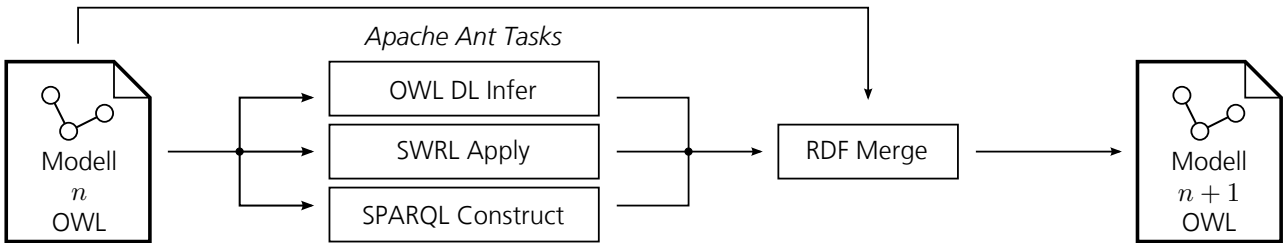

Abbildung 6-6: In der technischen Umsetzung wird eine Modellinstanz durch einen Inferenz- bzw. Transformationsschritt mittels OWL, SWRL oder SPARQL erweitert.

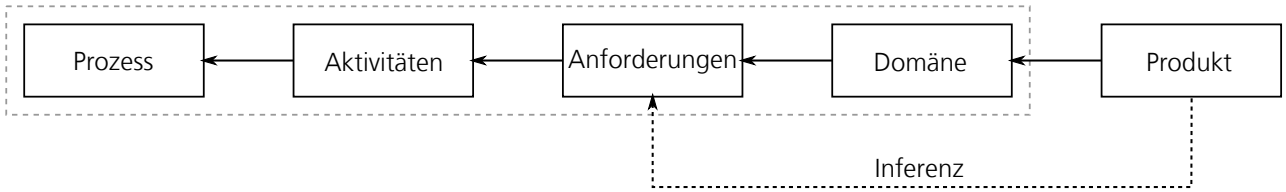

Abbildung 6-7: Für das Produkt wird durch die Modellierung der Anwendungsdomäne die Inferenz eines konkreten Prozessmodells möglich.

Axiomen geschlossen werden kann. Hier können sogenannte Schließungsaxiome verwendet werden, die den Lösungsraum weiter eingrenzen (schließen).

Abbildung 6-6 visualisiert die konkrete technische Realisierung, wie sie im Rahmen dieser Arbeit implementiert wurde. Auf Basis des Buildtools *Apache Ant* wurden die Komponenten *OWL DL Infer*, *SWRL Apply* und *SPARQL Construct* implementiert. Diese erschließen auf Basis eines Modells n implizites Wissen, und mittels der Komponente *RDF Merge* kann dieses mit vorhandenem Wissen kombiniert werden. Dabei entsteht das Modell $n+1$, was dann weiterverarbeitet werden kann.

6.5 Tailoring eines Prozessmodells

Als Resultat der Entdeckung impliziter Anforderungen durch den Einsatz einer Einflussanalyse ergeben sich Maßnahmen, die erfüllt werden müssen. Durch den Anwendungskontext dieser Maßnahmen und deren Umsetzung ergibt sich auch ein entsprechendes Vorgehen. In diesem Zusammenhang ist das Tailoring eines Vorgehensmodells aus den ermittelten Anforderungen der letzte Schritt zur Unterstützung eines einzelnen Entwicklers durch die Definition eines für ihn maßgeschneiderten Prozesses bzw. Workflows (siehe Abbildung 6-7). Dabei kann ein formalisiertes Vorgehensmodell in Zusammenhang mit einer Formalisierung von Methodenwissen verwendet werden.

So handelt es sich bei den aus den Normen zur Funktionalen Sicherheit abgeleiteten Prozessmodellen um generische Referenzprozesse, die auf den konkreten Bedarf im Unternehmen angepasst werden können. Dazu gehört etwa die Einhaltung von Empfehlungen bzgl. der Vorgehensweisen. Über diese muss jedes Unternehmen entscheiden, ob und wie es dabei vorgehen will. Beispielsweise können bestimmte Maßnahmen unterlassen werden, wenn dieses ausreichend und plausibel begründet wird. Somit taucht eine einzelne Maßnahme unter Umständen nicht bei der konkreten Entwicklung auf, sollte jedoch im Sicherheitsnachweis mit entsprechender Begründung enthalten sein.

Somit kommt nun die Verknüpfung zwischen Produkt- und Prozessmodellen zum Einsatz. Dazu sind die sich aus dem Einfluss eines ASIL ergebenden Anforderungen zum einen *produktbezogene* Anforderungen, aber auch *prozessbezogene* Anforderungen. Durch eine entsprechende Kategorisierung können solche Anforderungen mit bestimmten Phasen innerhalb

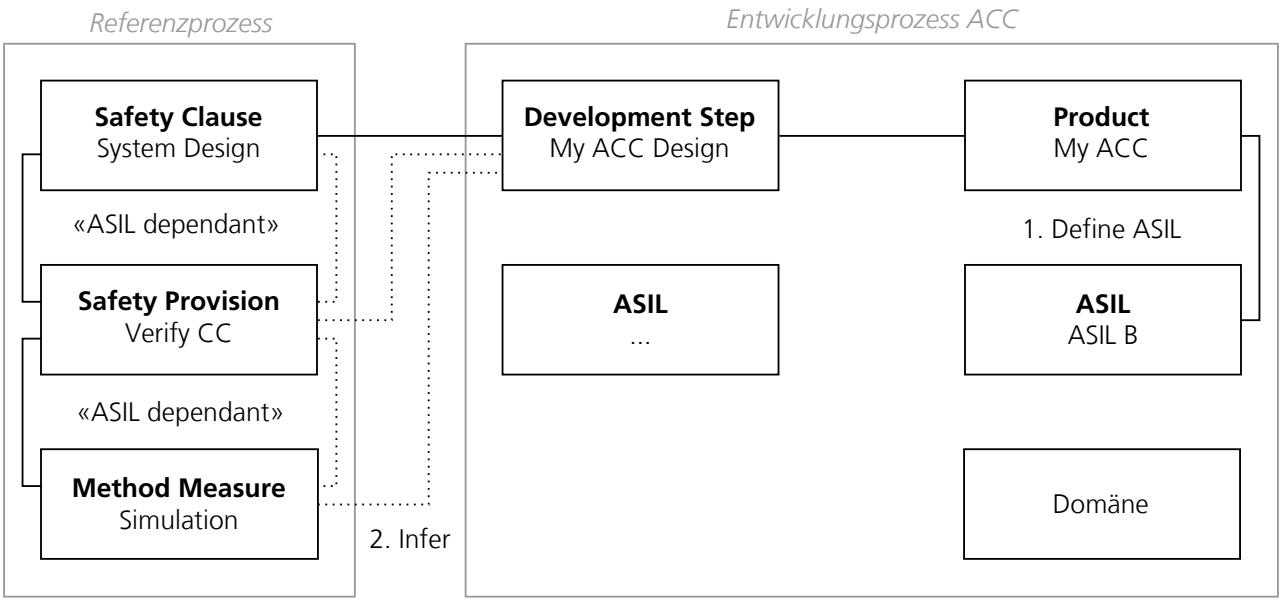

Abbildung 6-8: Eine beispielhafte Umsetzung des wissensbasierten Tailoring zur ISO DIS 26262.

eines Entwicklungsprozesses verknüpft werden. Auch hier kann eine Ontologie zum Einsatz kommen. So sind etwa in der ISO DIS 26262 Anforderungen mit bestimmten Entwicklungsphasen assoziiert.

Um die Konformität eines Unternehmensprozesses mit einem Referenzprozess nachzuweisen, kann das formalisierte Vorgehensmodell genutzt werden. Es umfasst notwendige Maßnahmen sowie Anforderungen und Prozessschritte. Insbesondere dadurch, dass sich über eine Eigenschaft der Domäne (hier dem ASIL) eine Verknüpfung zwischen dem Produkt und dem Referenzprozess ergibt, ist diese sehr einfach zu warten. Diese Verknüpfung (*Traceability Link*) ist eine *logische* Konsequenz eines Domänenattributes, und kann somit mittels eines Reasoners erschlossen werden. Abbildung 6-8 zeigt diese Anwendung am Beispiel der Entwurfsphase *System Design* nach ISO DIS 26262.

Bei der mathematischen Modellierung und Analyse der Überdeckung von Referenzprozess zu Unternehmensprozess können Eigenschaften der OWL ausgenutzt werden, insbesondere die Axiome der *Konzeptäquivalenz* und *Konzepthierarchie*. Wenn alle Entwicklungsschritte $c_{\mathrm{DevStep},i}$ in einem eigenen Prozessmodell für die Konformität zu einem Referenzprozess notwendig sind, so muss

$$\forall i : \{c_{\mathrm{DevStep},i}\} \sqsubseteq C_{\mathrm{RefStep},j} \vee \{c_{\mathrm{DevStep},i}\} \equiv C_{\mathrm{RefStep},j} \tag{6.8}$$

gelten. Dieses bedeutet, dass alle Entwicklungsschritte entweder Teil eines Referenzschritts $C_{\mathrm{RefStep},j}$ oder mit diesem identisch sind. Dieses schließt implizit den Fall ein, dass ein Entwicklungsschritt mehrere Referenzschritte abdeckt. Umgekehrt ist interessant, ob der eigene Prozess eine Umsetzung des Referenzprozesses

$$\forall j : \{c_{\mathrm{DevStep},i}\} \equiv C_{\mathrm{RefStep},j} \tag{6.9}$$

darstellt. Dazu müssen alle Referenzschritte durch eigene Entwicklungsschritte abgedeckt sein. Auch hier kann der Einsatz eines Reasoners unterstützen.

6.6 Verifikation der wissensbasierten Verwebung

Den in Kapitel 4 definierten Anforderungen wurde wie folgt begegnet. Für verständliche Anforderungen (Anforderung W1) wird ein zielorientiertes Requirements Engineering in Zusammenhang mit der Definition von Domänenontologien empfohlen. Dabei sind die verschiedenen

Modelle (GOMS, KAOS) über die Metamodellebene miteinander verknüpft, um *Kompatibilität* (Anforderung W4) zu gewährleisten. Diese lose gekoppelte Modellierung ermöglicht vollständige domänenspezifische Modelle, die zur *Konfliktlösung* und *Qualitätssicherung* (Anforderung W5) genutzt werden können. Zur *Formalisierung* (Anforderung W3) wird die OWL empfohlen. Diese deckt eine *offene* Formalisierung ab (Anforderung W2). Über das ODM kann eine Verknüpfung mit den technischeren Metamodellen erfolgen. Durch die formale Basis der OWL werden Analysen wie die gezeigte Einflussanalyse möglich. Bezüglich der *Traceability* (Anforderung W6) liefert eine wissensbasierte Modellierung die Möglichkeit zur Aufdeckung impliziter Abhängigkeiten. Hinzu kommt die Option Traceability Links nicht manuell zu warten, sondern als Resultat von Fakten in der Anwendungsdomäne zu schlussfolgern. Weiterhin wurde gezeigt, wie die Ableitung impliziter Anforderungen aus der Anwendungsdomäne genutzt werden kann, um eine *Handlungsplanung* (Anforderung W7) aus einem generischen Prozess maßzuschneidern (Tailoring).

6.7 Zusammenfassung des Kapitels

Es wurde gezeigt, wie eine Modellverwebung im Kontext modellgetriebener Entwicklung erfolgen kann. Dazu werden OWL Ontologien als formale Analysemodelle herangezogen. Gleichzeitig wurde beschrieben, wie sich Ontologien und Metamodelle im Kontext modellgetriebener Entwicklung ergänzen und deren konzeptionelle Basis bilden können. Aktuelle Standards (ODM) unterstützen diesen Trend und bieten Möglichkeiten zur technischen Integration.

Weiterhin wurde beschrieben, wie die Nutzung von Ontologien Möglichkeiten bietet, Analysen auf formalisierten Modellen durchzuführen. Dabei können insbesondere durch die Verknüpfung von Anforderungs- und Domänenmodellen implizite Anforderungen mittels Inferenz entdeckt werden. Dazu wurde eine Einflussanalyse formal modelliert. Zur Kompensation der Schwächen der Verwendung von OWL (Komplexität und Ausdruck) bietet sich die Möglichkeit der Verwendung ergänzender Beschreibungsmittel (SWRL, SPARQL) zur Modelltransformation anstelle des Schließens mittels Beschreibungslogik.

Die Verknüpfung eines Vorgehensmodells mit Domänenwissen kann genutzt werden, um einen konkreten Prozess anforderungsgemäß maßzuschneidern. Eine wissensbasierte Modellierung bietet weiterhin den Vorteil, dass Traceability Links aus Fakten in der Domäne geschlossen werden können und somit nicht mehr explizit verwaltet werden müssen.

Technisch wurden einzelne Komponenten einer generischen Werkzeugkette (basierend auf Apache Ant) implementiert. Diese ermöglichen logisches Schließen (OWL DL Inferenz), regelbasiertes Schließen (SWRL) und Abfragen (SPARQL) sowie das Zusammenführen von Ontologien (RDF). Auf Basis der OWL API wird der Reasoner HermiT verwendet. Auf RDF basierende Operationen nutzen das Jena Framework. Diese Komponenten können allgemein auf OWL Ontologien angewendet werden.

6.8 Zusammenfassung des Teils: Formale Verwebung interdisziplinärer Entwicklungsaktivitäten

Mit dem aktuellen Kapitel schließt auch der Teil zur formalen Verwebung interdisziplinärer Entwicklungsaktivitäten. Hiermit ist ein umfassendes Rahmenwerk definiert, welches einen formalisierten Entwicklungsprozess umfasst. Dieser bildet die Grundlage für eine verwobene Entwicklung verschiedener Fachdisziplinen. Abbildung 6-9 zeigt einen Überblick über die Struktur der gesamten Arbeit bis zu diesem Punkt.

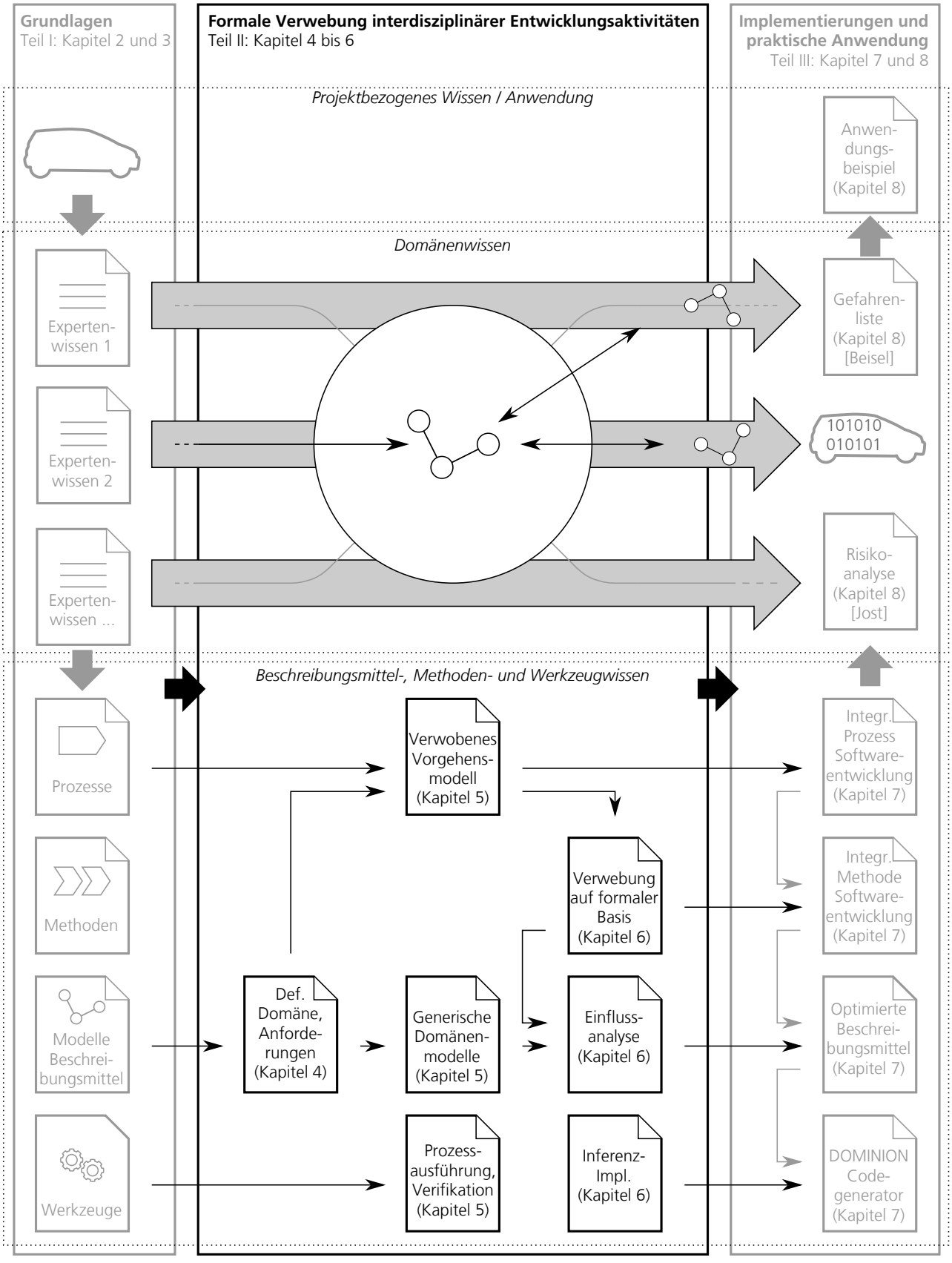

Abbildung 6-9: Struktur der Arbeit.

Die beschriebenen Methoden bilden die in großen Teilen domänenunabhängige Grundlage für eine effizientere und besser vernetzte Gesamtentwicklung. In einem ersten Schritt (Kapitel 4) wurde dazu die AAS Domäne mit den Entwicklungssträngen *Funktionsentwicklung*, *Funktionale Sicherheit* und *Faktor Mensch* definiert, sowie Anforderungen an den Gesamtentwicklungsprozess abgeleitet.

Darauf aufbauend wurden Modelle definiert, die den Entwicklungsprozess sowie grundlegende Domänenartefakte formal fassen (Kapitel 5), um maschinenunterstützte Analysen während der Entwicklung (wie in Kapitel 6) über Entwicklungsstränge hinweg überhaupt erst möglich zu machen. Hierzu zählt vor allem die Einflussanalyse eigener Entwurfsentscheidungen im Kontext der Gesamtentwicklung. Diese nutzt Modelle der Anwendungsdomäne, sowie die beschriebenen Produkt- und Prozessmodelle. Umgesetzt wurden diese in OWL. Die genannten Analysemöglichkeiten stellen einen wesentlichen Vorteil des in dieser Arbeit beschriebenen Ansatzes dar, sie vernetzen verschiedene Entwicklungsstränge und deren Wissen mit dem Ziel einer effizienteren Gesamtentwicklung.

Darüber hinaus wurden prototypische Werkzeugentwicklungen durchgeführt, um schnell neue Analysemodelle in eine Gesamtwerkzeugkette integrieren zu können. Hierzu zählt unter anderem ein Werkzeug zur Inferenz (Schlussfolgerung). Für das formalisierte Vorgehensmodell wurde weiterhin ein Werkzeug entwickelt, um es in ein ausführbares Prozessmodell (BPEL) zu überführen sowie über den Modelchecker NuSMV zu verifizieren.

Im folgenden Teil wird auf die Konkretisierung der OWL Modelle in ausführbare Softwareumgebungen eingegangen. Des Weiteren wird eine beispielhafte Anwendung beschrieben, die ebenfalls prototypisch umgesetzt wurde.

Teil III

Implementierungen und praktische Anwendung

7 Implementierung von Assistenz- und Automationssystemen

7.1 Ziele des Kapitels

Nach der Beschreibung des Vorgehensmodells und insbesondere der Verwebung der Entwicklungsstränge im vorigen Kapitel steht in diesem Kapitel die Verknüpfung mit den Implementierungen im Mittelpunkt. Dieses umfasst einen konkreten Vorschlag, wie die abstrakten konzeptuellen Modelle (Ontologien) mit konkreten Implementierungen interagieren können. Die vorliegenden Implementierungen beziehen sich vornehmlich auf Software, die dazugehörigen Aktivitäten sind also dem Entwicklungsstrang *Funktionsentwicklung* sowie dessen Teilentwicklungssträngen zugeordnet.

Zunächst werden in Abschnitt 7.2 wichtige Trends und Anforderungen im Kontext der Implementierung künftiger Assistenz- und Automationssysteme erläutert. Daraufhin werden in Abschnitt 7.3 und in Abschnitt 7.4 beschrieben, wie diese Technologien in der AAS Domäne Automobil angewendet werden können. Dieses umfasst den Aspekt der Modellierung von Implementierungen ebenso wie deren Instanziierung in einer Laufzeitumgebung. Dabei spielt die Serviceorientierung eine zentrale Rolle, siehe auch [GHHK08, KGH08, GHH08, NSM+08, RGS+08a, RGS08b, RKS+08, SHG+10].

7.2 Softwareentwicklung im Automobil

Bei Betrachtung der Entwicklung von Rechnerhardware im Automobil lässt sich erkennen, dass die Verfügbarkeit von Rechnerkapazitäten [Wei09] in einem Abstand von acht Jahren der eines gewöhnlichen PCs folgt. Bei der Software ist ein ähnlicher Trend auszumachen. Dabei erwartet man beispielsweise für 2012 bis zu einem Gigabyte Binärcode in neu entwickelten Automobilen. Dieses entspricht einer Komplexität, die etwa fünf Jahre Verzug gegenüber Desktop PCs aufweist [BKPS07].

Für das Software Engineering bzw. die Software lässt sich hieraus leicht ableiten, dass mit einer ähnlichen Verzögerung auch die Möglichkeiten der Softwareentwicklung folgen. Setzte man in den 1980er Jahren etwa kaum Betriebssysteme ein, so ist inzwischen Hardware-Abstraktion und der Einsatz eines Betriebssystems üblich. Dieses findet sich auch in der *AUTomotive Open System ARchitecture* (AUTOSAR) [FBH+06] wieder. Eine besonders ausgeprägte Abstraktion ist bei AUTOSAR in dem virtuellen Funktionsbus zu erkennen, welcher Software-Komponenten miteinander verbindet und eine hohe Flexibilität und Austauschbarkeit dieser Komponenten während der Entwicklung gewährleisten soll. Zur Laufzeit ist diese Flexibilität jedoch stark eingeschränkt.

Der hohe Bedarf nach Flexibilität zur Entwicklungszeit bei AUTOSAR hat betriebswirtschaftliche Gründe. Die Entwicklung eines Automobils ist stark dezentralisiert und somit sind viele verschiedene Unternehmen an der Entwicklung eines neuen Automobils unter der Marke eines Automobilherstellers beteiligt. Durch die Schaffung eines offenen Standards wie AUTOSAR erlangen Automobilhersteller mehr Möglichkeiten, Software (und damit deren Zulieferer) auszutauschen. Umgekehrt sind die Markteintrittsbarrieren für einen Zulieferer bei einem Automobilhersteller ebenfalls kleiner geworden.

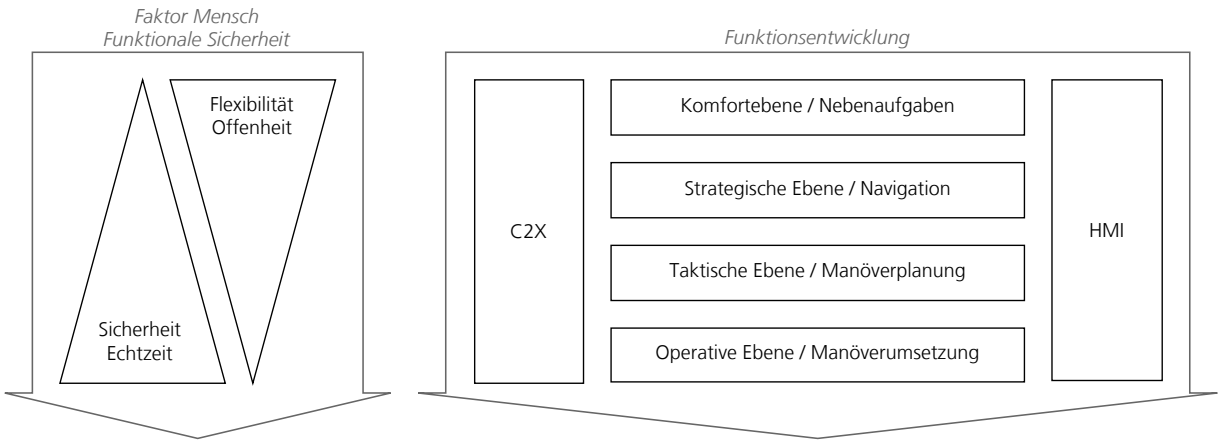

Abbildung 7-1: Bei modernen Implementierungen handelt es sich um komplexe Mehrebenensysteme im Spannungsfeld äußerer Anforderungen von *Offenheit*, *Flexibilität* und *Sicherheit*.

Extrapoliert man diesen Trend zur Virtualisierung und betrachtet die Entwicklung bei Softwaresystemen auf dem PC oder im Internet, so lässt sich ableiten, dass auch im Automobil eine hohe Flexibilität zur Laufzeit zur Verfügung stehen wird. Solche Middleware Technologien ermöglichen es, einzelne Anwendungen soweit von der Hardware zu abstrahieren, dass diese dynamisch zur Laufzeit umverteilt werden können. Dieses ist dann sinnvoll, wenn es die der Domäne entsprechenden Echtzeitanforderungen zulassen. Weiterhin ist zu beachten, dass bestimmte Anwendungen an eine bestimmte Hardware gebunden sind. Dieses betrifft insbesondere solche Anwendungen, die direkt mit Sensoren oder Aktoren interagieren.

Der betriebs- und volkswirtschaftliche Bedarf resultiert insbesondere durch die aktuellen Initiativen, Fahrzeuge untereinander (Car2Car) sowie mit einer Verkehrsinfrastruktur (Car2Infrastructure) zu vernetzen. Durch den Einsatz dieser neuen Kommunikationsinfrastruktur bzw. aus den daraus möglichen neuen verkehrlichen Anwendungen [CAR07, Eur09] soll vor allem ein positiver Einfluss auf Verkehrssicherheit und -effizienz erzielt werden. Betriebswirtschaftlich interessant sind jedoch vor allem die Möglichkeiten im Bereich Infotainment (Internet Anwendungen) und Kundendienst (Ferndiagnose und Wartung von Software). Gerade die beiden letztgenannten ermöglichen neue Geschäftsmodelle, vergleichbar mit denen aus der Mobilkommunikation. Öffnet man das Fahrzeug, so ist es wahrscheinlich, dass individualisierte AAS entwickelt werden können.

Technisch impliziert dieses Vorgehen eine *Öffnung* der Fahrzeugarchitektur nach außen. Es handelt sich somit nicht mehr um ein isoliertes Stück statischer Software, sondern um ein *komplexes Mehrebenensystem* innerhalb einer weitläufigen und komplexen *Anwendungslandschaft* (siehe Abbildung 7-1). Somit ändert sich die Aufgabe eines Softwareentwicklers und -architekten. Weniger stark steht die Neuentwicklung von Software im Vordergrund, vielmehr ist es die *Integration* neuer Komponenten und bereits existierender Software.

Im Bereich von Business Software bzw. Internet Anwendungen sind diese Herausforderungen schon länger vorhanden. Unter Betonung der Integration und stärkerer Ausrichtung an unternehmensübergreifenden Geschäftsprozessen entstand das Konzept der *Service-Oriented Architecture* (SOA). Zentral ist hierbei die Kapselung von Funktionen in Diensten. Diese stellen unter Verwendung einer standardisierten semiformalen Beschreibung Informationen über ihre Schnittstellen, Operationen und Sichtbarkeit bzw. Erreichbarkeit bereit. Solche Dienste werden zu höherwertigen Diensten durch die Modellierung von Geschäftsprozessen *orchestriert*, siehe Abbildung 7-2. Auch orchestrierte Dienste können wiederum in Geschäftsprozessen verwendet werden. Aus diesem Architekturkonzept entstand eine Reihe von offenen Standards, u.a. beim

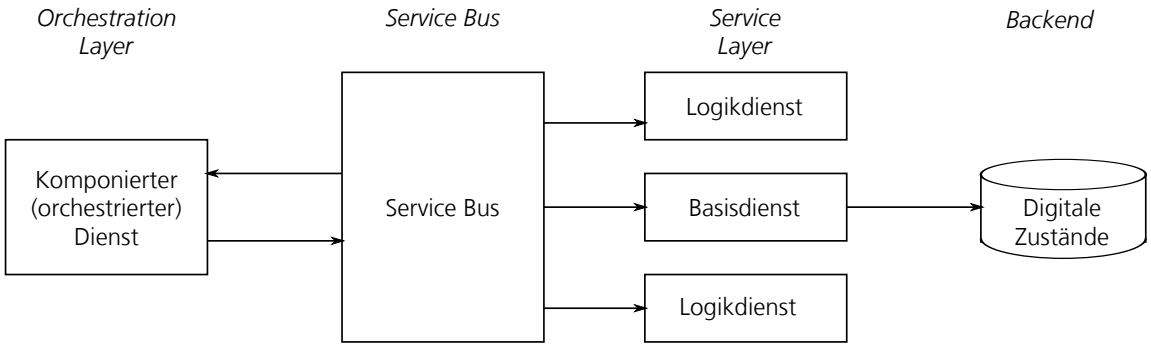

Abbildung 7-2: Existierende Dienste können auf der prozessbezogenen Orchestrierungsebene zu neuen Diensten komponiert werden.

World Wide Web Consortium (W3C) und bei der *Organization for the Advancement of Structured Information Standards* (OASIS).

Entwicklungen im Umfeld des Web 2.0, z.B. das *Cloud Computing*, setzen auf Konzepten von SOA auf. Des Weiteren existieren akademische Arbeiten, die sich mit der formalen Modellierung von SOA Diensten beschäftigen [Aga07].

Neben der erwähnten *Offenheit* wird bei SOA die *Flexibilität* dadurch gewährleistet, dass es neben der offenen Beschreibung eines Dienstes Standards gibt, die Datenübertragungsprotokolle auf Basis von HTTP bereitstellen: *Simple Object Access Protocol* (SOAP)[BEK+00]. Somit kann ein Dienst sehr einfach durch einen ähnlichen Dienst ersetzt werden. Besonders vor dem Hintergrund der Entwicklungen im Bereich *Faktor Mensch* ist diese Flexibilität wichtig. Vor allem zu frühen Entwicklungsphasen müssen schnell und kostengünstig verschiedene HMI Konzeptalternativen unter Verwendung technischer Prototypen evaluiert werden können. Dabei ist die erwähnte *Offenheit* eine wichtige Basis, um *Flexibilität* überhaupt zu ermöglichen.

Eine weitere wichtige Anforderung ist die *Sicherheit*. Neben Maßnahmen für den Prozess fordern aktuelle Standards zur Entwicklung sicherheitskritischer Systeme Maßnahmen zur Einhaltung und Nachweis der funktionalen Sicherheit. Die Forderung nach Sicherheit wird jedoch mit existierenden Implementierungen nicht ausreichend adressiert. Zwar existieren Standards [LK06] zur Sicherheit im Sinne der Datensicherheit (*security*), jedoch nicht zur funktionalen Sicherheit (*safety*) von SOA Software. Des Weiteren eignen sich die existierenden Implementierungen von Ausführungsumgebungen und Datenübertragungsprotokollen (SOAP) nicht zur Ausführung von echtzeitkritischen Funktionen. Für die beiden wesentlichen Kritikpunkte

- Sprachstandards (siehe Abschnitt 7.3)
- Laufzeitumgebung (siehe Abschnitt 7.4)

werden in den folgenden Abschnitten Lösungen für den Einsatz in echtzeitkritischen Umgebungen vorgeschlagen.

7.3 In-Vehicle Service Languages

Ein Beispiel für die Anpassung einer verbreiteten und generischen Programmiersprache an die Bedürfnisse der Entwicklung sicherheits- und echtzeitkritischer Systeme ist MISRA C [McC04]. Diese definiert eine vereinfachte Untermenge der Programmiersprache C.

Die Aufgabe für die in diesem Abschnitt vorgeschlagenen *In-Vehicle Service Languages* ist ungleich einfacher. So kann in diesem Fall ein verändertes Metamodell mittels XML Schema einfach definiert und konkrete Modelle gegen die XML Schema Spezifikation maschinell geprüft

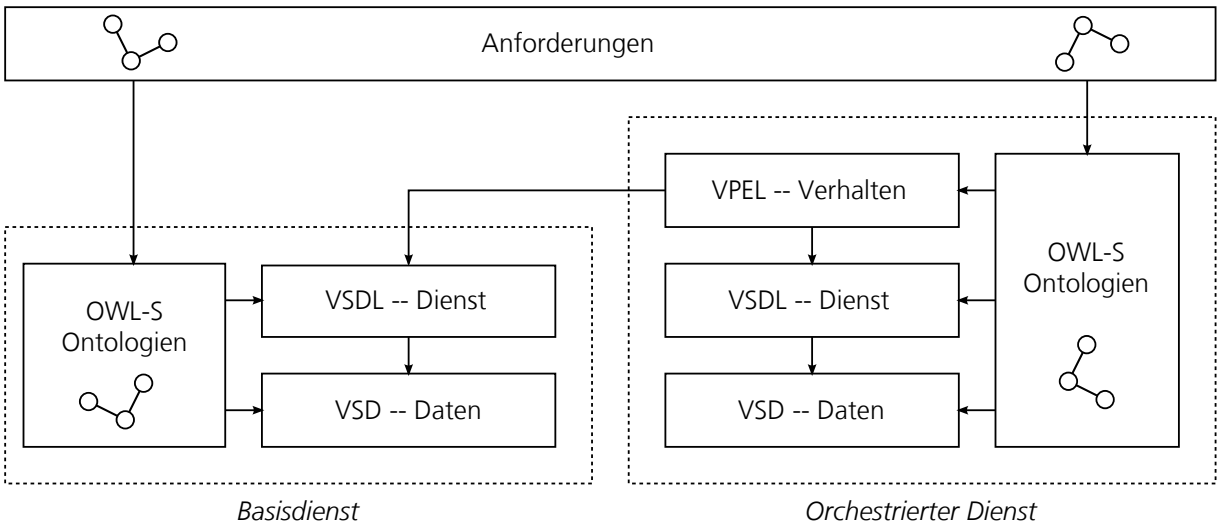

Abbildung 7-3: Die *In-Vehicle Service Languages* beschreiben den Dienst (VSDL), seine Datenmodelle (VSD), sowie ggf. sein Verhalten (VPEL).

werden. Die Teile der *In-Vehicle Service Language* können im Rahmen von modellbasierter Entwicklung eingesetzt werden. Weiterhin sind die größten Einschränkungen von C (dynamische Speicherverwaltung, Zeiger) gar nicht Teil der *In-Vehicle Service Languages*. Im Folgenden werden die wichtigsten Elemente der *In-Vehicle Service Languages*

- *In-Vehicle Schema Definitions* (VSD)
- *In-Vehicle Service Description Language* (VSDL)
- *In-Vehicle Process Execution Language* (VPEL)

beschrieben und bzgl. ihrer Eignung für die Entwicklung sicherheits- und echtzeitkritischer Systeme bewertet. Sie werden im Rahmen dieser Arbeit aus den offenen Standards XSD, WSDL und BPEL abgeleitet.

In Abbildung 7-3 sind jeweils ein Basisdienst und ein orchestrierter Dienst dargestellt. VPEL spezifiziert ggf. das Verhalten, VSDL die Schnittstellen des Dienstes an sich und VSD umfasst die Beschreibung der Daten, die zwischen den Diensten ausgetauscht werden. Dabei kann nun die in den Grundlagen beschriebene OWL-S verwendet werden, um mittels ihres *Groundings* eine Verbindung zwischen in Ontologien beschriebenen Anforderungen und der Spezifikation eines Dienstes zu übernehmen.

In-Vehicle Schema Definitions (VSD)

Die VSD (In-Vehicle Schema Definition) lässt sich aus dem *eXtensible Schema Definition* (XSD) Standard des W3C [FW04] ableiten. Die XSD stellt ein Metamodell für die Beschreibung von XML Dialekten und somit zur Beschreibung baumartiger Datenstrukturen zur Verfügung.

Vorteile

XSD erlaubt neben der Modellierung konkreter Strukturen [TBMM04] (komplex und einfach) und Attribute auch die Definition eigener komplexer und einfacher Datentypen [BM04], die dann beispielsweise eine Teilmenge natürlicher oder kontinuierlicher Zahlen bildet. Der Sprachumfang ist ausreichend zur Beschreibung statischer Datenstrukturen, wie sie in eingebetteten Systemen üblich sind.

Nachteile

Die Ausdrucksmöglichkeiten der XSD gehen an einigen Stellen für eingebettete (speicherbegrenzte) und echtzeitkritische (zeitbegrenzte) Systeme zu weit. Dabei ist stellvertretend zu nennen, dass die Angabe der Anzahl von auftretenden Instanzen eines bestimmten Datentyps nicht verpflichtend ist. Ebenso die Schnittmenge (`xsd:oneOf`, Äquivalent zu `union` in C) und Vereinigungsmenge (`xsd:allOf`, kein Äquivalent in C) sind auch nach MISRA C zumindest nicht empfohlen. Des Weiteren sind nicht alle primitiven Datentypen geeignet.

Beschreibung VSD (Primitive Datentypen)

Aus XSD abgeleitet, kann ein Datentyp T^*_{VSD} zunächst als Menge

$$T^*_{\mathrm{VSD}} = \{v_{\mathrm{VSD}}, l_{\mathrm{VSD}}, f_{\mathrm{VSD}}\} \tag{7.1}$$

definiert werden, bestehend aus dem Werteraum v_{VSD} und der lexikalischen Darstellung l_{VSD}. Die Facetten f_{VSD} beschreiben Eigenschaften des Werteraums. Entscheidend für die Eignung in Richtung echtzeitkritischer Systeme ist die Beschreibung der Facetten. XSD definiert die *fundamentalen* Facetten *Gleichheit*, *Ordinalität*, *Beschränktheit*, *Kardinalität* und *Numerizität*. Zur Einschränkung des Werteraums v_{VSD} existieren die *nicht-fundamentalen* Facetten `length`, `minLength`, `maxLength`, `pattern`, `enumeration`, `whiteSpace`, `maxInclusive`, `maxExclusive`, `minExclusive`, `minInclusive`, `totalDigits` und `fractionDigits`.

Grundsätzlich ist zu fordern, dass bezüglich der *Kardinalität* Endlichkeit erfüllt sein muss, um eine Abbildung auf ein deterministisch arbeitendes Echtzeitsystem zu gewährleisten. Handelt es sich um numerische Datentypen, ist außerdem Beschränktheit zu fordern. Diese Eigenschaften können inhärent durch den Werteraum v_{VSD} erfüllt sein. Ist dieses nicht der Fall, so müssen diese Eigenschaften durch nicht-fundamentale Facetten hergestellt werden.

Inhärent geeignete primitive Datentypen sind `float`, `double` und `boolean`. Geeignete abgeleitete Datentypen sind `long`, `int`, `short`, `byte`.

Im Hinblick auf VSDL und den Einsatz im Bereich eingebetteter Systeme ist diese Menge zu erweitern zu

$$T_{\mathrm{VSD}} = \{v_{\mathrm{VSD}}, l_{\mathrm{VSD}}, f_{\mathrm{VSD}}, b_{\mathrm{VSD}}\} \tag{7.2}$$

wobei b_{VSD} einer binären Darstellung entspricht. Hierbei kann allerdings auf Standards zur Darstellung von Fließkommazahlen [IEE85] und Ganzzahlen [McC04] verwiesen werden. Im Rahmen einer statischen Implementierung können die in der VSD beschriebenen Daten in C/C++ Datenstrukturen umgesetzt und binär repräsentiert werden. Bei Ausführung in einer flexiblen serviceorientierten Implementierung kann wieder die lexikalische Darstellung aus der XSD verwendet werden.

Beschreibung VSD (Komplexe Datentypen)

Der zweite Teil von XSD beschäftigt sich mit den Datenstrukturen, welche baumartig aufgebaut sind. Des Weiteren können Baumknoten attributiert werden. Jedoch lassen sich die in Attributen enthaltenen Informationen auch in der Baumstruktur eines XML Dokuments repräsentieren. Daher werden Attribute nicht weiter in der VSD berücksichtigt. Diese Vereinfachung reduziert vor allem auch den Implementierungsaufwand, um VSD technisch zu verarbeiten. Komplexe Datenstrukturen setzen sich aus weiteren komplexen Datenstrukturen und einfachen Datentypen zusammen. Auch bei der Betrachtung der Datenstrukturen kann wieder die MISRA C zur Orientierung herangezogen werden.

Eine komplexe Datenstruktur kann durch den Einsatz von geordneten Tupeln (Sequenz) weitere Strukturen enthalten. Dabei muss jedes beschriebene Element in der vorgegebenen Reihenfolge vorkommen und findet sein Äquivalent in `struct` in der Programmiersprache C. Mit der Menge

T_{VSD} aller Datentypen und der Menge S aller Datenstrukturen S_{VSD} kann formal für die Strukturen

$$S_{\mathrm{VSD}} = (S_0, \ldots, S_n) \text{ mit } S_i \subseteq (S \cup T_{\mathrm{VSD}}) \text{ und } 0 \leq i \leq n \text{ und } n > 0 \tag{7.3}$$

gefordert werden. Eine Datenstruktur S_{VSD} ist ein geordnetes Tupel, welches sich aus Vektoren von elementaren Datentypen T_{VSD} und anderen Datenstrukturen S_{VSD} zusammensetzt.

Des Weiteren sind viele Deklarationen in der XSD optional, d. h. die Instanziierung eines Elements muss nicht durchgeführt werden bzw. die Anzahl an Knoten ist beliebig. Dieses muss eingeschränkt werden. Auch dafür existieren in der XSD Attribute zur Beschreibung der Multiplizität von Knoten.

In-Vehicle Service Description Language (VSDL)

Die VSDL (In-Vehicle Service Description Language) lässt sich aus der WSDL (Web Services Description Language) ableiten. Die WSDL stellt die semiformale Beschreibung von Schnittstellen, Datenmodellen (mittels XSD), Operationen und Adressierung der Implementierung eines Dienstes zur Verfügung.

Vorteile WSDL

WSDL ist sehr verbreitet und enthält die wesentlichen Elemente, um die Funktionalität eines Dienstes in einer eigenen Implementierung bzw. Geschäftsprozess zu nutzen. Der Sprachumfang reicht aus, um auch statische Softwaremodule zu beschreiben.

Nachteile WSDL

Da die WSDL selber nur Schnittstellen und nicht Datenstrukturen oder Verhalten beschreibt, liegen keine Nachteile im Sinne des Echtzeitverhaltens vor. Daher kann die WSDL als VSDL übernommen werden.

Beschreibung VSDL

Ein in VSDL formalisierter Dienst lässt sich als Menge

$$S_{\mathrm{VSDL}} = \{t_{\mathrm{VSDL}}, m_{\mathrm{VSDL}}, e_{\mathrm{VSDL}}, b_{\mathrm{VSDL}}, p_{\mathrm{VSDL}}, s_{\mathrm{VSDL}}\} \tag{7.4}$$

beschreiben. Es besteht aus den folgenden Elementen.

- *Types* t_{VSDL}: Dieser Teil umfasst die mit VSD beschriebenen Datentypen.
- *Messages* m_{VSDL}: Innerhalb der Definition einer Nachricht werden Datentypen aggregiert. Diese können dann bei der Ausführung als Eingangs- und Ausgangsnachrichten instanziiert werden.
- *Port Types* e_{VSDL}: Hier werden Typen von Kommunikationsendpunkten beschrieben, welche Operationen sowie assoziierte Eingangs- und Ausgangsnachrichten umfassen.
- *Binding* b_{VSDL}: Dabei wird für einen bestimmten Typen von Kommunikationsendpunkt Operationen und Nachrichten an ein konkretes Protokoll (z.B. SOAP) gebunden. Hier können auch eigene (neue) Protokolle eingebunden werden.
- *Port* p_{VSDL}: Ein bestimmter Typ von Kommunikationsendpunkt wird instanziiert.
- *Service* s_{VSDL}: Ein Dienst aggregiert mehrere Ports (Kommunikationsendpunkte).

In-Vehicle Process Execution Language (VPEL)

Die VPEL (In-Vehicle Process Execution Language) leitet sich von BPEL (Business Process Execution Language) ab. BPEL war bis zur Version 1.1 ein Industriestandard, ist ab Version 2.0 über OASIS ebenfalls ein offen diskutierter Standard [JE07]. BPEL deckt den Bereich der Orchestrierung von Diensten ab, dabei entsteht ein abstrakterer, komponierter Dienst.

Vorteile BPEL

BPEL ist sehr umfangreich betreffend die Modellierung von Verhalten. Es erlaubt die bedingte Abarbeitung von Prozessen. Dabei können Dienste mit ihrer Funktionalität aufgerufen werden. Aktionen können parallel und seriell erfolgen. Auch Interaktion mit Menschen (z.B. per E-Mail) ist Teil der Syntax, was daraus resultiert, dass BPEL ursprünglich aus der Modellierung von Geschäftsprozessen entstanden ist. Gerade diese Elemente können eine Grundlage zur Modellierung von komfortorientierten Fahrerassistenzsystemen bilden, wie etwa Kommunikations- oder auch Mautabrechnungsdiensten.

Nachteile BPEL

BPEL enthält einige Elemente zur Verhaltensmodellierung, die im echtzeitkritischen Bereich nicht zu empfehlen sind. Dazu gehören, in Anlehnung an MISRA C, vor allem Schleifen ohne explizite Endbedingung.

Beschreibung VPEL

Das wichtigste Kriterium bei der Bewertung einer Ablaufsprache wie BPEL ist die *Echtzeitfähigkeit*. Dies bedeutet, dass ein Ablauf und dessen Elemente in endlicher Zeit abgearbeitet werden können. Ein weiterer wichtiger Aspekt ist die *Erreichbarkeit*. Diese bedeutet, dass es keine unerreichbaren Zustände geben darf. Ein Prozess

$$P_{\mathrm{VPEL}} = \{p_{\mathrm{VPEL}}, d_{\mathrm{VPEL}}, b_{\mathrm{VPEL}}, a_{\mathrm{VPEL}}, s_{\mathrm{VPEL}}\} \tag{7.5}$$

nach VPEL beinhaltet die folgenden Elemente.

- *Partner Link* p_{VPEL}: Diese Elemente verknüpfen den VPEL Prozess mit VSDL Dienstebeschreibungen. Dies gilt sowohl für das Konsumieren als auch Anbieten von Diensten.
- *Data Handling* d_{VPEL}: Im Rahmen eines VPEL Prozesses können mit VSD deklarierte Daten instanziiert werden. Dabei können die Werte von Variablen anderen Instanzen zugewiesen werden. Es sind verschiedene *Query Languages* erlaubt. Verwendet man XPath [MM99] (eine Abfragesprache zur Adressierung von Teilen eines XML Dokumentes), so ist dies bei Beachtung der Einschränkungen von VSD unkritisch.
- *Basic Activities* b_{VPEL}: Basisaktivitäten umfassen vor allem Wertzuweisung, sowie synchrones und asynchrones Aufrufen externer Dienste sowie Fehlerbehandlung. Bis auf die `wait` Aktivität sind diese Aktivitäten unkritisch, bei `wait` ist auf Einhaltung der Echtzeitfähigkeit zu achten.
- *Structured Activities* a_{VPEL}: Strukturierte Aktivitäten umfassen Elemente für serielle und parallele Abläufe, sowie für sich wiederholende und konditionale Aktivitäten. Kritisch bezüglich der Echtzeitfähigkeit ist die Definition von Abbruchkriterien bei sich wiederholenden Aktivitäten. Bezüglich der Erreichbarkeit ist die Definition bedingter Aktivitäten kritisch.
- *Scopes* s_{VPEL}: Diese dienen vor allem zur Kapselung von Aktivitäten und zum Bereitstellen verschiedener Kontexte.

Zwei wichtige Aspekte von BPEL werden für VPEL nicht berücksichtigt.

- *Variable Properties*: Diese werden zur expliziten Annotation von Variablen genutzt.

- *Correlation*: Betrifft die Behandlung von mehrfacher Instanziierung von Geschäftsprozessen und die Zuordnung einzelner Nachrichten zu einer bestimmten Instanz, unter Verwendung von *Variable Properties*.

7.4 Laufzeitumgebung

Aus dem in Abschnitt 7.2 beschriebenen Mehrebenensystem ergeben sich unterschiedliche Anforderungen an die zu verwendenden Laufzeitumgebungen. Hierzu werden im Weiteren drei verschiedene Laufzeitumgebungen beschrieben.

1. *Komfortebene / Navigation*: Hier liegen wenige oder keine Echtzeitanforderungen vor, Flexibilität ist sehr wichtig. Auf dieser Ebene bietet sich der Einsatz eines regulären *Enterprise Service Bus* (ESB) an. Dieser entspricht einer regulären SOA Infrastruktur.

2. *Manöverebene*: Diese beinhaltet den Einsatz einer leichtgewichtigen Laufzeitumgebung, die eine effiziente Kommunikation und Steuerung von Implementierungen ermöglicht, die *weiche* Echtzeitanforderungen erfüllen müssen. Im Rahmen dieser Arbeit kommt die DOMINION Laufzeitumgebung zum Einsatz (siehe Abschnitt 7.4.1).

3. *Stabilisierungsebene*: Konzeptionell ist ebenso eine Laufzeitumgebung zur Ausführung solcher Komponenten vorgesehen, die *harten* Echtzeitanforderungen genügen müssen und auf *Electronic Control Units* (ECU) instanziiert werden. Dieses umfasst vor allem sicherheitskritische Komponenten sowie Basissoftware zum Betrieb von Sensoren und Aktoren.

Trotz verschiedener Implementierungen herrscht für alle diese Komponenten dieselbe konzeptionelle Sicht. Zum Zeitpunkt des *Entwurfs* kann die volle Flexibilität einer SOA Lösung genutzt werden. Jedoch besitzt die Implementierung je nach *Deployment* eine andere Flexibilität bei ihrer *Instanziierung*. Genau dieses Deployment schließt die Lücke zwischen Dienstemodell (Abschnitt 7.3) und ihrer Instanziierung in einer Laufzeitumgebung, wie sie in diesem Abschnitt beschrieben sind. Sie stellt sicher, dass eine Implementierung automatisiert in verschiedenen Laufzeitsystemen verfügbar ist. Des Weiteren muss dieser Mechanismus derart konstruiert sein, dass Dienste über die Grenzen einer Laufzeitumgebung hinweg interagieren bzw. orchestriert werden können.

Eine Übersicht der verschiedenen Ausprägungen der Laufzeitumgebungen sind in Abbildung 7-4 zu erkennen. Neben dem Einsatz eines klassischen ESB für Internetdienste sind zwei weitere Laufzeitumgebungen (DOMINION und die ECU Ebene) beteiligt, die sich jedoch aus Sicht der Dienste nur in ihrer Implementierung unterscheiden.

7.4.1 DOMINION Laufzeitumgebung

Die DOMINION [GHH08] Laufzeitumgebung zielt darauf ab, die Anforderungen für *weiche* Echtzeit zu erfüllen. Ziel ist dabei der Einsatz moderner Multitasking Betriebssysteme. Dabei garantiert Ubuntu[1] Linux mit präemptiver Echtzeiterweiterung bei entsprechender Priorisierung der Prozesse Latenzzeiten weit unterhalb einer Millisekunde. Die Interprozesskommunikation erfolgt lokal über *Memory Mapped I/O* (MMIO) und verteilt über *User Datagram Protocol* (UDP) Netzwerkkommunikation. Der Vorteil solcher modernen Betriebssysteme (anstelle von Steuergeräten) ist, dass sie sehr einfach in hochdynamischen Umgebungen (Internetanwendungen) eingebunden werden können, genauso wie zur Feldbuskommunikation mit eingebetteten Systemen.

[1] Ubuntu Linux: `http://www.ubuntu.com`

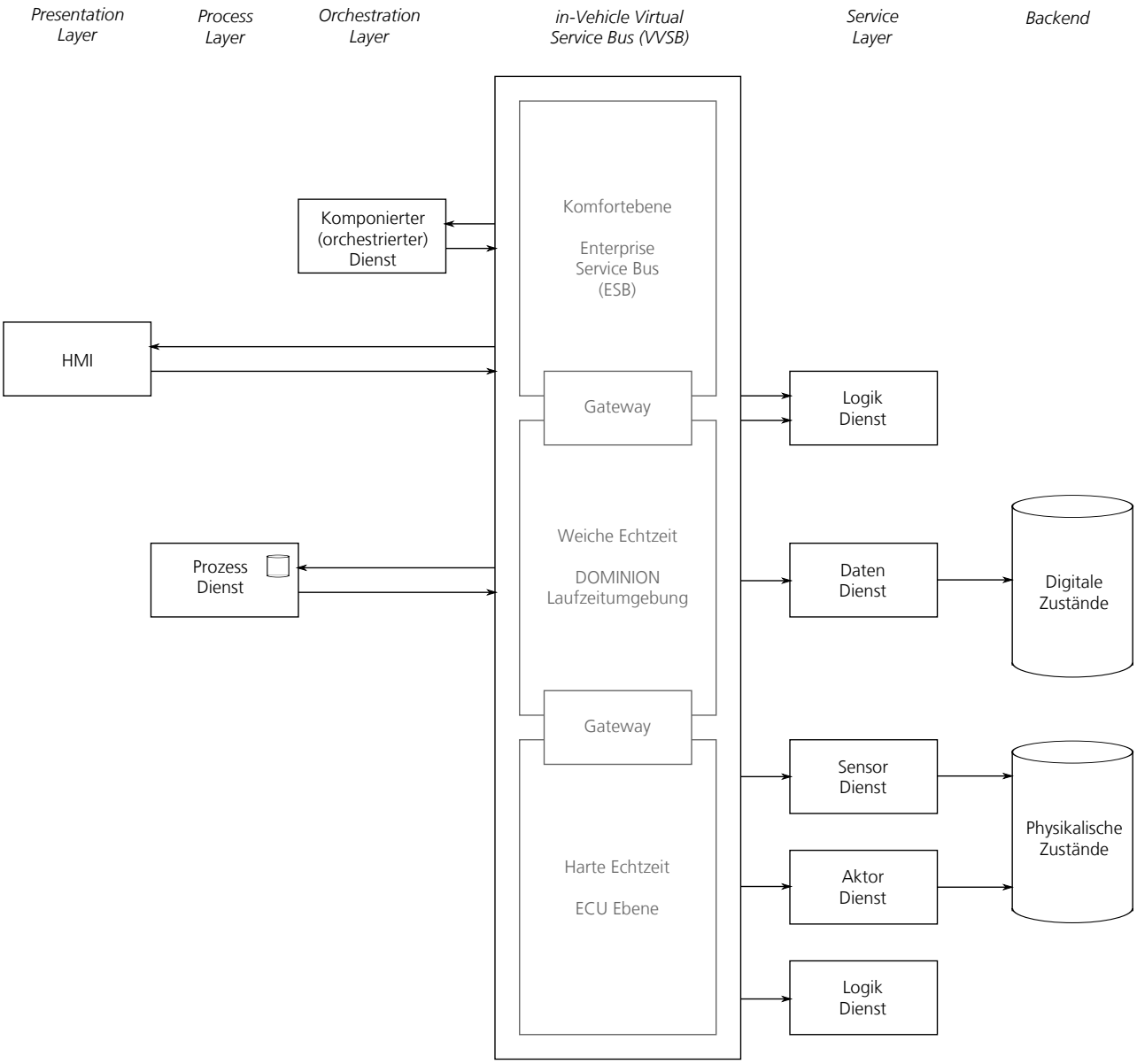

Abbildung 7-4: Entsprechend des Charakters von Mehrebenensystemen werden drei verschiedene Laufzeitumgebungen in einem konsistenten Dienstemodell integriert.

7.4.2 Laufzeitanalyse: Deployment

Um die Flexibilität während des Entwicklungsprozesses gewährleisten zu können, bietet das SOA Konzept den Vorteil, dass getrennt von der Implementierung eines Dienstes eine semiformale Beschreibung dessen vorliegt. Bei Verwendung von BPEL liegt sogar das Verhalten in einem maschinenlesbaren Format vor. Diese Trennung bietet die Möglichkeit, die Integration I der Diensteimplementierungen mittels einer Modelltransformation t (Codegenerierung)

$$I = t(P_{\mathrm{VPEL}}, D_{\mathrm{VSDL}}, S_{\mathrm{VSD}}) \tag{7.6}$$

auf verschiedene Laufzeitumgebungen abzubilden. Bei Ausführung des Dienstes in einem BPEL Container ist nicht einmal eine solche Transformation notwendig, da der Container als Interpreter wirkt.

Dieses Vorgehen bietet weiterhin den Vorteil, dass das Deployment auf verschiedene Laufzeitumgebungen Teil eines Migrationskonzepts sein kann. Dieses ist solange notwendig, bis Ausführungscontainer zur Verfügung stehen, die leistungsstark genug sind, um die Echtzeitanforderungen zu erfüllen. Dass die Einführung solcher Konzepte wahrscheinlich wird, liegt an der starken Parallelisierung im Entwurf von Rechnerstrukturen (Mehrprozessorsysteme, für 2022 werden mehr als 1000 Prozessorelemente für tragbare Rechner prognostiziert [Int08]). Je abstrakter und präziser eine Sprache definiert ist, desto besser können ihre Sprachelemente durch eine technische Optimierung automatisch parallelisiert werden.

7.4.3 Laufzeitsynthese: Hybrid Services

Durch das Deployment sind die Diensteimplementierungen innerhalb ihrer speziellen Laufzeitumgebungen integriert. Da es sich bei Fahrerassistenzsystemen jedoch um ein *komplexes Mehrebenensystem* handelt, müssen die Komponenten über die Grenzen der Laufzeitumgebungen hinweg interagieren können. Dafür gibt es verschiedene Ansätze. Eine Möglichkeit, die bei [Häg08] diskutiert wird, ist die Einrichtung eines zentralen Gateways zwischen den Laufzeitumgebungen. Vorteil ist die Bündelung der Interaktion zwischen den Laufzeitumgebungen, was jedoch zum Flaschenhals werden kann. Darüber hinaus unterstützt der Ansatz von [Häg08] weder WSDL noch SOAP.

Im Weiteren wird daher ein anderer Ansatz verwendet, der der *HybridServices*. Dabei existieren Dienste, deren Implementierung technisch parallel auf zwei Laufzeitumgebungen zurückgreifen kann. Somit besteht keine Notwendigkeit für ein Gateway, da jeder Dienst für sich genommen technisch zu einem Gateway werden kann. Dieses wird möglich, indem verschiedene Ports mit verschiedenen Operationen eines einzelnen Dienstes an unterschiedliche Laufzeitumgebungen gebunden werden. Dieser Ansatz hat den weiteren Vorteil, dass das Konzept der Architektur konsistent bleibt, da das Gateway nur auf der Implementierungsebene zu finden ist, nicht jedoch Teil des Konzepts wird.

7.4.4 Technische Umsetzung

Um das Konzept zu validieren und für den Aufbau von Demonstratoren nutzen zu können, wurde ein Codegenerator mittels XSLT umgesetzt, der die Modelltransformation aus den In-Vehicle Sprachen in Quellcode bzw. API übersetzt. Zur Demonstration der Synthese bzw. hybriden Dienste wurde die Codegenerierung prototypisch für DOMINION so angepasst, dass diese ebenso über SOAP in einem interpretierten VPEL Prozess in einen ESB eingebunden werden können.

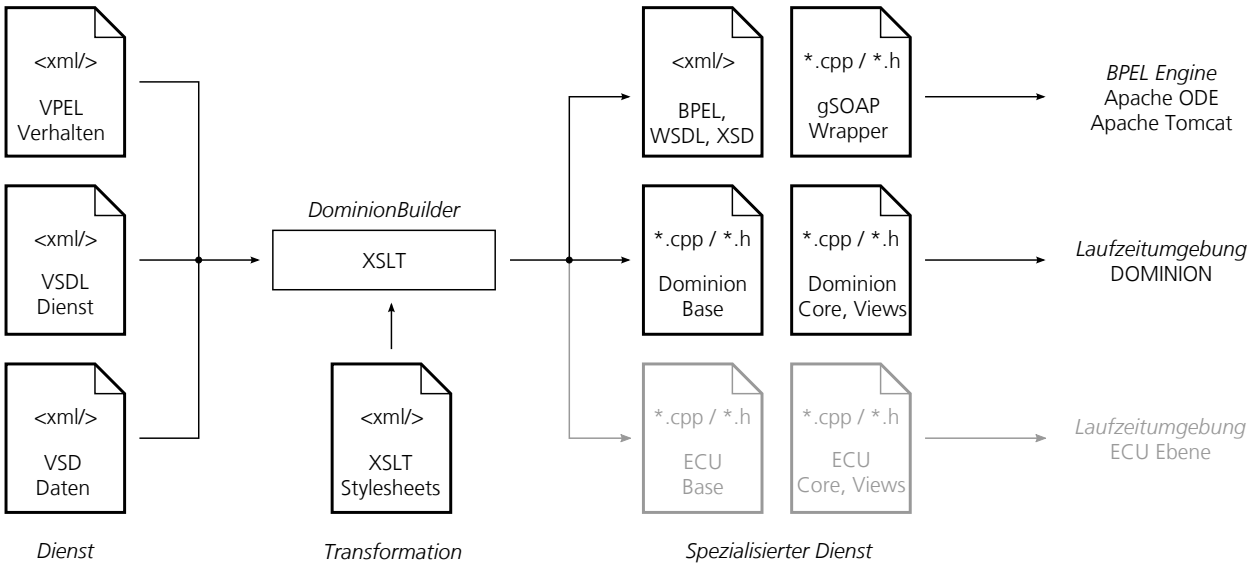

Abbildung 7-5: Bei dem Schritt des *Deployment* wird ein Dienst für seinen Laufzeitkontext spezialisiert. Prototypisch wurde dieses Vorgehen für den ESB und die DOMINION Laufzeitumgebung umgesetzt.

Dazu wurde die Bibliothek gSOAP[2] verwendet. Zur Orchestrierung von Diensten kam Apache ODE[3] auf Apache Tomcat[4] zum Einsatz.

Für die DOMINION Laufzeitumgebung werden die semiformalen XML Modelle mittels XSLT in C/C++ Quelltext übersetzt, der die vorliegende Implementierung mit der jeweils speziellen API der Laufzeitumgebung verbindet. Dabei werden die Datenstrukturen und Datentypen als geschachtelte Datenstrukturen in C/C++ umgesetzt, siehe auch Abbildung 7-5.

7.5 Reglerstruktur / Softwarearchitektur ACC

Die Auswirkungen der Serviceorientierung auf den Funktionsentwurf im Kontext einer komplexen Gesamtentwicklung lassen sich an einem Beispiel verdeutlichen. Der wichtigste Einfluss durch einen serviceorientierten Entwurf ergibt sich aus der Zustandslosigkeit eines Dienstes. Diese unterstützt eine lose Kopplung insbesondere dadurch, dass es keine impliziten Annahmen gibt, die nicht in der Spezifikation des Dienstes enthalten sind. Die wichtigste Annahme bei zeitdiskreten Reglern als Softwarekomponente liegt in der Spezifikation der Abtastrate, also der Annahme, dass eine Funktion regelmäßig in einem konstanten zeitlichen Raster aufgerufen wird. Die Benutzung und Wiederverwendung einer solchen Funktion beeinflusst also maßgeblich ihre Funktionsweise, was in der Praxis schnell fehleranfällig ist.

Eine serviceorientierte Spezifikation eines solchen Reglers verbietet solche impliziten Annahmen über den Anwendungskontext. Durch diese strikte Bedingung wird die Aufgabe der Integration deutlich vereinfacht. Exemplarisch wird dabei eine mögliche Realisierung eines ACC Systems im aktiven Zustand (Regelung von Abstand und Geschwindigkeit) beschrieben, siehe dazu Abbildung 7-6.

Durch die Verwendung von Schätzungen der Position und Geschwindigkeit des vorausfahrenden Fahrzeugs können die beiden Regler für Abstand und Geschwindigkeit rein als Proportionalregler ausgeführt werden. Sie umfassen also keinen dynamischen Zustand. In diesem Fall kommt die

[2] gSOAP: `http://www.cs.fsu.edu/~engelen/soap.html`
[3] Apache ODE: `http://ode.apache.org/`
[4] Apache Tomcat: `http://tomcat.apache.org/`

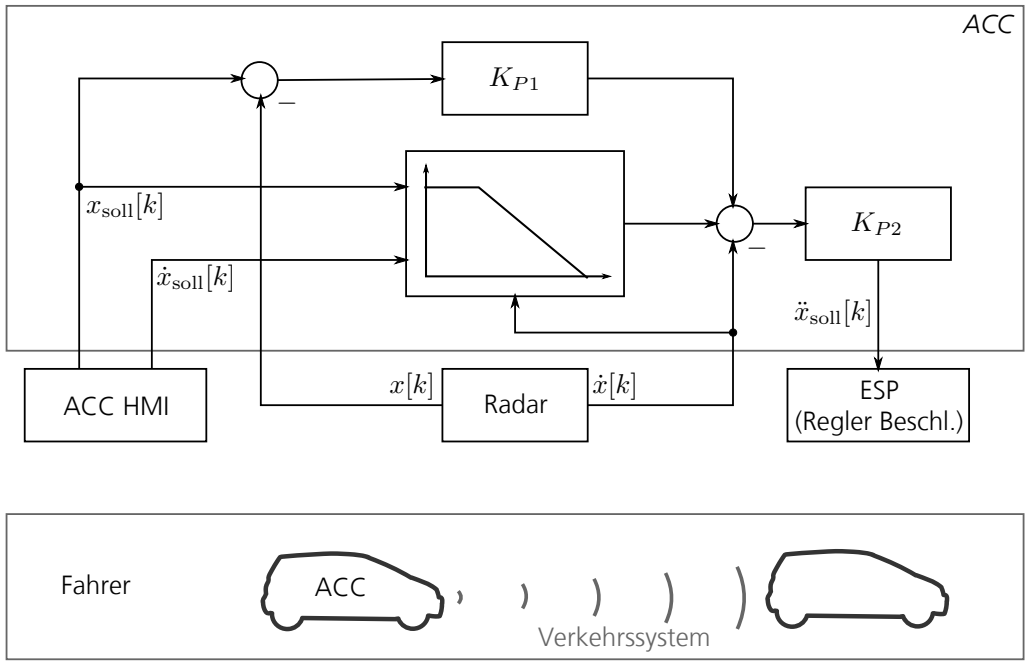

Abbildung 7-6: Der ACC Regler, dargestellt als Blockschaltbild.

Statik nicht nur der Softwareintegration, sondern auch der Stabilität und Reaktivität des Systems zugute.

Das hier realisierte ACC besteht aus drei wichtigen Komponenten. Ein unterlagerter Regler K_{P2} für die Geschwindigkeit $x[k]$ setzt auf einen basalen Beschleunigungsregler auf. Darauf setzt die Annäherungsstrategie auf. Dabei handelt es sich um eine statische nichtlineare Funktion. Diese begrenzt ab einem bestimmten Abstand die Sollgeschwindigkeit $\dot{x}_{\text{soll}}[k]$ basierend auf der Geschwindigkeit des vorausfahrenden Fahrzeugs. Hinzu kommt ein proportionaler Regler K_{P1} für den Abstand $x[k]$, der eine bleibende Regelabweichung des Abstands verhindert.

In Abbildung 7-7 ist die serviceorientierte Umsetzung des ACC zu erkennen. Wichtig ist die zustandslose Umsetzung der Dienste, die Sensordaten liefern bzw. Aktoren betätigen. In Analogie zu Datendiensten liefern die Sensor- und Aktordienste Zugriff auf eine Persistenzschicht. Handelt es sich bei klassischen SOA Lösungen üblicherweise um eine Datenbank, so handelt es sich bei der skizzierten Umsetzung um physikalische Zustände.

7.6 Zusammenfassung des Kapitels

Das vorliegende Kapitel verdeutlicht, wie die abstrakten konzeptuellen Modelle, wie sie in Kapitel 5 und Kapitel 6 eingeführt wurden, in der Funktionsentwicklung genutzt werden können und diese unterstützen. Dabei wird eine Lösung vorgeschlagen, die wie in den Kapiteln zuvor Entwicklungen aus dem Bereich des Internet aufgreift und somit insbesondere Offenheit und Flexibilität erhöht. Konkret handelt es sich um die OWL-S, eine OWL Ontologie, die über eine technische Bindung auf der WSDL aufsetzt. Somit können Beschreibungsmittel aus dem serviceorientierten Umfeld (SOA) aufgegriffen werden. Da diese Beschreibungsmittel jedoch für Internetanwendungen optimiert sind, wurde ein Ansatz aufgezeigt, der durch eine Modifikation existierender Standards diese Beschreibungsmittel für echtzeitkritische AAS über In-Vehicle Services nutzbar macht. Dazu wurden geeignete Teilmengen der WSDL, XSD und BPEL definiert. Weiterhin ist die Serviceorientierung ein Ansatz für Softwarearchitekturen, der weiter geht als komponentenorientierte Ansätze und somit aktuelle Softwaretechnologie in die AAS Domäne einbringt.

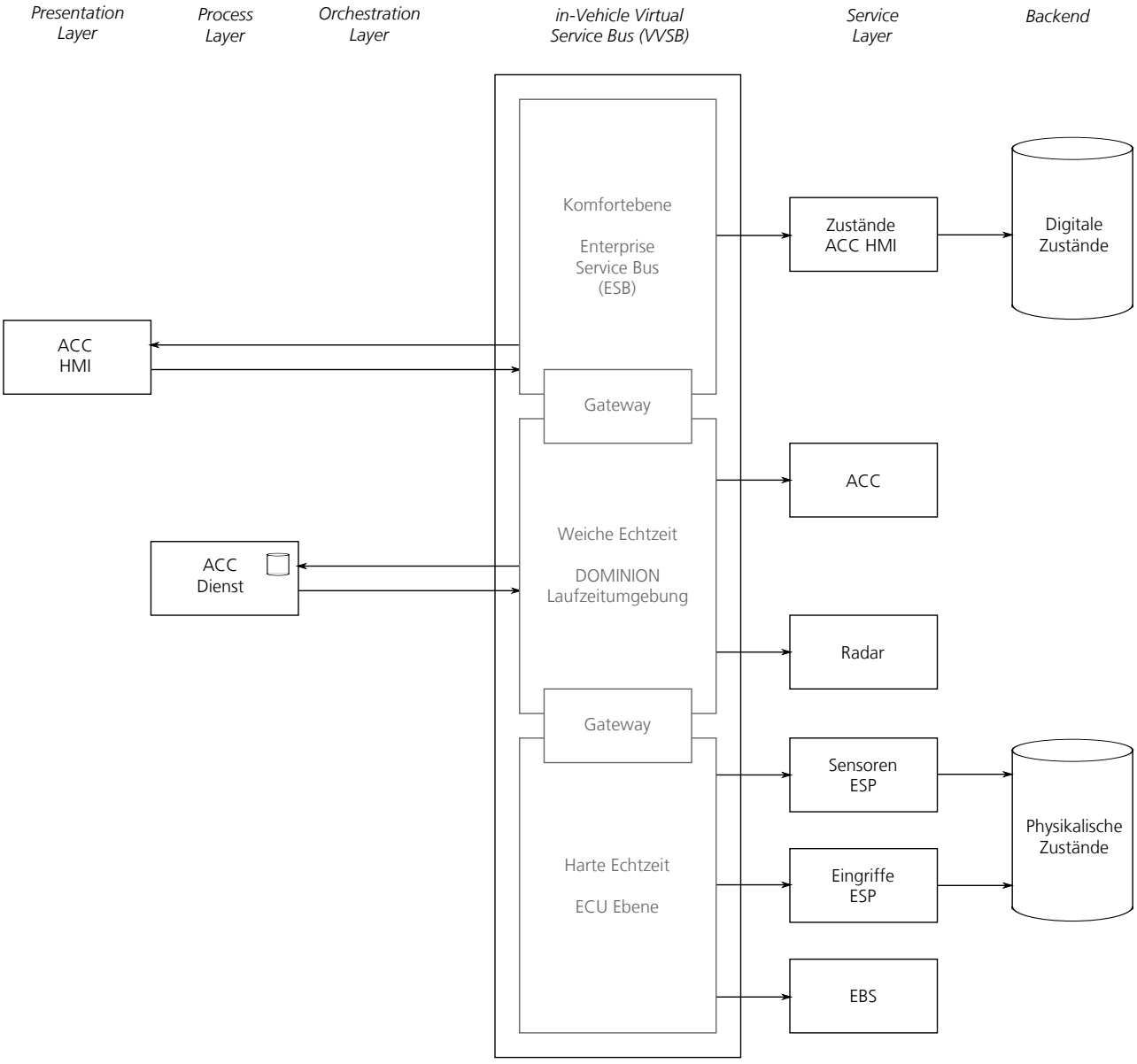

Abbildung 7-7: Das ACC, modelliert nach dem SOA Paradigma.

Die beschriebene Technologie wurde prototypisch implementiert, inklusive eines Codegenerators für die DOMINION Laufzeitumgebung für laufzeit- und echtzeitkritische AAS. Der Einsatz eines Codegenerators, einem zentralen Element modellgetriebener Ansätze, ermöglicht eine effiziente technische Implementierung generischer sowie abstrakter Modellelemente. Die Verwendung eines serviceorienterten Paradigmas hat Auswirkungen auf die Funktionsentwicklung, insbesondere durch die Forderung der Zustandslosigkeit eines Dienstes. Diese kann zu einem erhöhten Implementierungsaufwand führen. Jedoch führt die Zustandslosigkeit zu einfacher Wiederverwendung, was insbesondere bei komplexen, interdisziplinären Entwürfen von Vorteil ist.

Die Codegenerierung basiert in diesem Fall auf XSLT, dabei werden aus den Elementen der VSD Datenstrukturen generiert. Die VSDL wird auf einen Wrapper für die DOMINION Laufzeitumgebung abgebildet. Aus VPEL werden die konkreten Verknüpfungen zwischen den Datenschnittstellen einzelner Dienste abgeleitet. Abhängig von der Spezifikation der technischen Bindung in der VSDL kann ein hybrider Dienst erzeugt werden, der eine DOMINION Laufzeitumgebung mit SOAP Diensten verbindet.

8 Anwendungsbeispiel

8.1 Ziele des Kapitels

Dieses Kapitel greift das in Abschnitt 4.3 eingeführte und bereits immer wieder in Teilen dargelegte ACC mit Notbremsassistenzsystem auf (siehe auch [Gac10]). Dazu wird zunächst in Abschnitt 8.2 das in Abschnitt 6.3 beschriebene Vorgehen in eine generische Werkzeugkette abgebildet und somit weiter konkretisiert. Dabei wird die Verwebung zwischen *Funktionsentwicklung* und *Funktionaler Sicherheit* auf Basis der ISO DIS 26262 betont. Als Anwendungsbeispiel wurde die in Abschnitt 4 beschriebene Anwendung eines ACC mit Notbremsassistenzsystem EBS verwendet.

Weiterhin wird in Abschnitt 8.3 das bereits in Abschnitt 4.3 eingeführte Systemmodell aufgegriffen. Dazu gehört ein Domänenmodell, welches in einer automatisierten Risikoanalyse genutzt wird. Weiterhin kommen Ziel- und Anforderungsmodelle nach den GOMS und KAOS Methoden zum Einsatz, sowie Modelle für eine Einflussanalyse. Diese wiederum sind mit einer serviceorientierten Implementierung verknüpft.

Als Ausblick erfolgt in Abschnitt 8.4 die Implementierung der statischen Logik des Notbremssystems in OWL. Dabei wird skizziert, wie hier insbesondere *Offenheit*, *Flexibilität* und *Sicherheit* auf der Implementierungsebene zusammenspielen und somit eine Basis für Individualisierung und Evolution von Assistenz- und Automationssystemen bilden können.

Weitere Bilder von der Umsetzung des beschriebenen Anwendungsbeispiels finden sich in Abschnitt C im Anhang dieser Arbeit.

8.2 Werkzeugkette und eine Iteration ACC / EBS

Beispielhaft wird eine einzelne Iteration einer Entwicklung für ein sicherheitskritisches AAS beschrieben: das EBS, ein Notbremsassistent. Dabei wird eine Iteration des Entwurfs innerhalb des Entwicklungsstrangs *Funktionsentwicklung* skizziert. Zu Beginn dieser Iteration liegt bereits ein ACC System vor, vergleiche auch Abschnitt 4.3. Das Gesamtsystem soll nun um ein EBS für außerstädtische Anwendungen erweitert werden. Während des Entwurfs und der technischen Umsetzung soll nun bewertet werden, ob es sinnvoll ist, das ACC und ein EBS auf Basis des existierenden Radarsensors aufzubauen. Solch eine Beschreibung und Definition einer Änderung funktionaler Anforderungen ist der erste Schritt einer solchen Iteration in der Entwicklung. Bei dieser Beschreibung wird das bis dahin modellierte Domänenwissen verwendet. Mit der Funktionsbeschreibung hängt eine erste Bewertung der Gefahren und des vom neuen Gesamtsystem ausgehenden Risikos zusammen. Aufgrund der hohen Gefahr, die durch eine versehentlich ausgelöste Bremsung ausgeht, könnte die Funktion *Gefahrenbremsung* mit einem hohen ASIL (B, C oder D) bewertet werden. In der Architektur des ACC Systems (siehe Abbildung 8-1) war das ACC mit ASIL A das sicherheitskritischste Teilsystem. Ein hoher ASIL des EBS hätte somit nicht Auswirkungen auf bereits existierende Komponenten, insbesondere auf die Diagnose des Radars.

> „Was muss getan werden, um eine ausreichende Sicherheit für das EBS sicherzustellen? Und was bedeutet die Einführung des EBS für die existierenden Komponenten des ACC?"

Zur Beantwortung dieser Fragen werden die in dieser Arbeit beschriebenen Methoden genutzt. Zu deren Nutzung wurden die einzelnen Implementierungen der Methoden in einer prototypischen

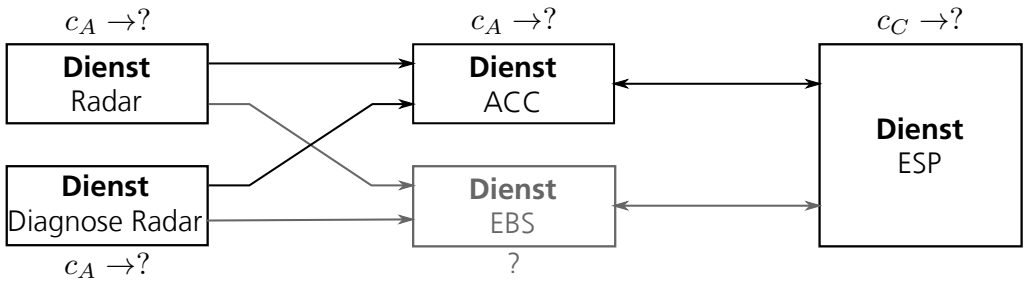

Abbildung 8-1: Das Notbremssystem EBS wird Teil der existierenden Systemarchitektur des ACC und hat somit Einfluss auf die Sicherheitseigenschaften des Gesamtsystems.

Werkzeugkette integriert. Für die Werkzeugkette ergibt sich ein bestimmter Ablauf. Die einzelnen Schritte sind entweder per XSLT oder Reasoner (OWL DL, SWRL, SPARQL) automatisiert (siehe dazu auch Abbildung 8-2, bzw. im Anhang Abbildung C-1 und Abbildung C-2). Sie lauten generisch wie folgt:

1. *Funktionale Anforderungsdefinition*: Dieser Schritt erfolgt mit Hilfe des Werkzeugs Objectiver, welches die KAOS Methode umsetzt. Dabei wird die Funktion und ihre Systemgrenzen über Domänenattribute annotiert. Das Objectiver Modell liegt in XML vor und wird anschließend in eine adäquate KAOS Ontologie transformiert.

2. *Analyse der Domäneneigenschaften*: Es werden für das System relevante Kombinationen von *exposure rate* E, *severity* S und *controllability* C über eine Analyse der Domänenattribute ermittelt.

3. *Bestimmung des ASIL*: Dieser ergibt sich aus den Faktoren E, S und C. Dabei werden Gefahren, die von einem System ausgehen, aus einer generischen Liste abgeleitet (*Automotive Generic Hazard List* [BRS10]). Somit kann eine Abschätzung des Ergebnisses einer Risikoanalyse erfolgen. Aus Domänenattributen werden verknüpfte Risiken identifiziert und entsprechende Faktoren E, S und C abgeleitet.

4. *Einflussanalyse*: Dabei wird ermittelt, wie sich die Änderung oder Einfügung einzelner Funktionen auf das komplette System auswirken.

5. *Tailoring*: Ableitung konkreter Anforderungen, Methoden, Maßnahmen und Prozessschritte aus dem integrierten Referenzprozess. In diesem Fall wird die ISO DIS 26262 verwendet. Dabei wird das formalisierte Modell der Norm genutzt, siehe auch [Jos10].

6. *Transformation*: Umwandlung in ein Vorgehensmodell nach Kapitel 5 mit dazugehörigem XML Modell, sowie der generierten und maßgeschneiderten Version der nun notwendigen Prozessschritte und Dokumentation.

In Abbildung 8-3 wird deutlich, wie sich diese analytischen Ontologien in die in Abschnitt 5 beschriebenen Ontologien einpassen.

- *AAS Domain*: Diese Ontologie umfasst die Konzepte, aus der sich die Domäne zusammensetzt, inklusive der generischen Konzepte, die für eine Risikoanalyse wichtig sind: Fahrsituation, Umgebungsbedingungen, etc.

- *AAS Product*: Diese Ontologie umfasst Konzepte, die speziell für das neue System bzw. Produkt sind. Über diese Ontologie findet eine Verknüpfung mit den Produktanforderungen statt.

- *AGHL* [Jos10]: Die *Automotive Generic Hazard List* (AGHL) wurde aus Unfall- und Systemcharakteristiken mit Blick auf Gebrauch und vorhersehbaren Missbrauch des Systems ermittelt. Dabei sind kausale Zusammenhänge zwischen Schaden und technischer

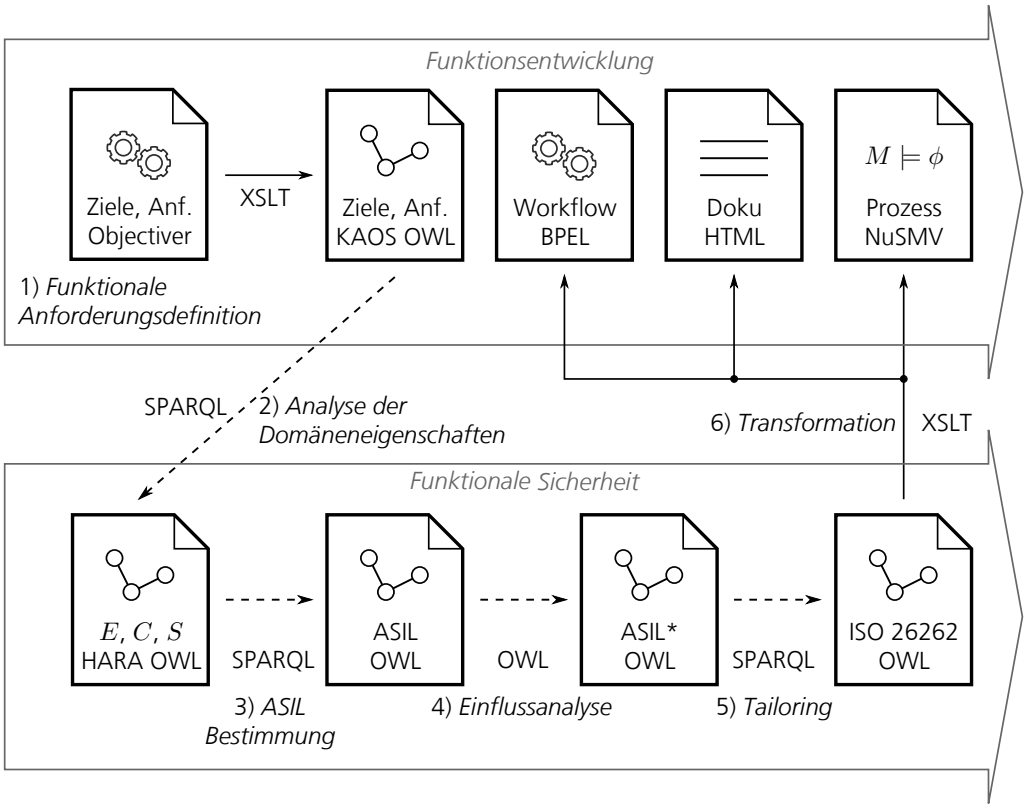

Abbildung 8-2: In der prototypischen Werkzeugkette werden zwei Entwicklungsstränge miteinander verwoben, dabei handelt es sich um die *Funktionsentwicklung* und um die *Funktionale Sicherheit*.

Ursache erfasst. Somit können bestimmte Eingriffe eines AAS bezüglich der von ihnen ausgehenden Gefahr klassifiziert werden.

- *HARA* [Jos10]: Diese Ontologie zu *Hazard Analysis and Risk Assessment* (HARA) nutzt den Reasoner dazu, um die der ISO DIS 26262 Risikoanalyse zugrundeliegende Tabellenlogik zu implementieren und automatisiert den ASIL zu bestimmen.

Die Werkzeugkette wurde dabei in die offene Entwicklungsplattform Eclipse[1] integriert. Wie in Kapitel 6 eingeführt, wurden generische Transformationsschritte für OWL, SWRL oder SPARQL in der Programmiersprache Java entwickelt. Als Reasoner wurde HermiT[2] eingesetzt. Insbesondere die Werkzeugkette wurde mit Apache Ant[3] umgesetzt.

Anzumerken an der technischen Implementierung ist insbesondere die Größe einzelner OWL Dateien, bestehend aus verschiedenen Axiomen. Insbesondere die Ontologie zur ISO DIS 26262 umfasst mehr als 1000 OWL Klasseninstanzen (und damit allein über 1000 Subklassenaxiome, hinzu kommen konkrete Relationen der einzelnen Instanzen). Dabei kommt die hohe Zahl vor allem durch die hohe Anzahl von Prozess- und Produktanforderungen zustande. Durch die hohe Anzahl an Knoten wird die Komplexität für den Reasoner sehr hoch, er würde auf einem aktuellen Rechner mehrere Stunden für eine Konsistenzprüfung bzw. die Inferenz neuen Wissens benötigen. Dieses ist jedoch kein Grund für einen Technologiewechsel, weg von der OWL. Aussagen, die direkt in OWL DL ausgedrückt werden können, können auch regelbasiert (SWRL) oder abfragebasiert (SPARQL) formuliert werden. Ab OWL 2 können auch die Profile EL, RL und QL verwendet werden. Die Verfügbarkeit dieser zusätzlichen Sprachstandards ist ein Vorteil des standardisierten Beschreibungsmittels OWL.

[1] Eclipse: http://www.eclipse.org/
[2] Hermit OWL Reasoner: http://hermit-reasoner.com/
[3] Apache Ant: http://ant.apache.org/

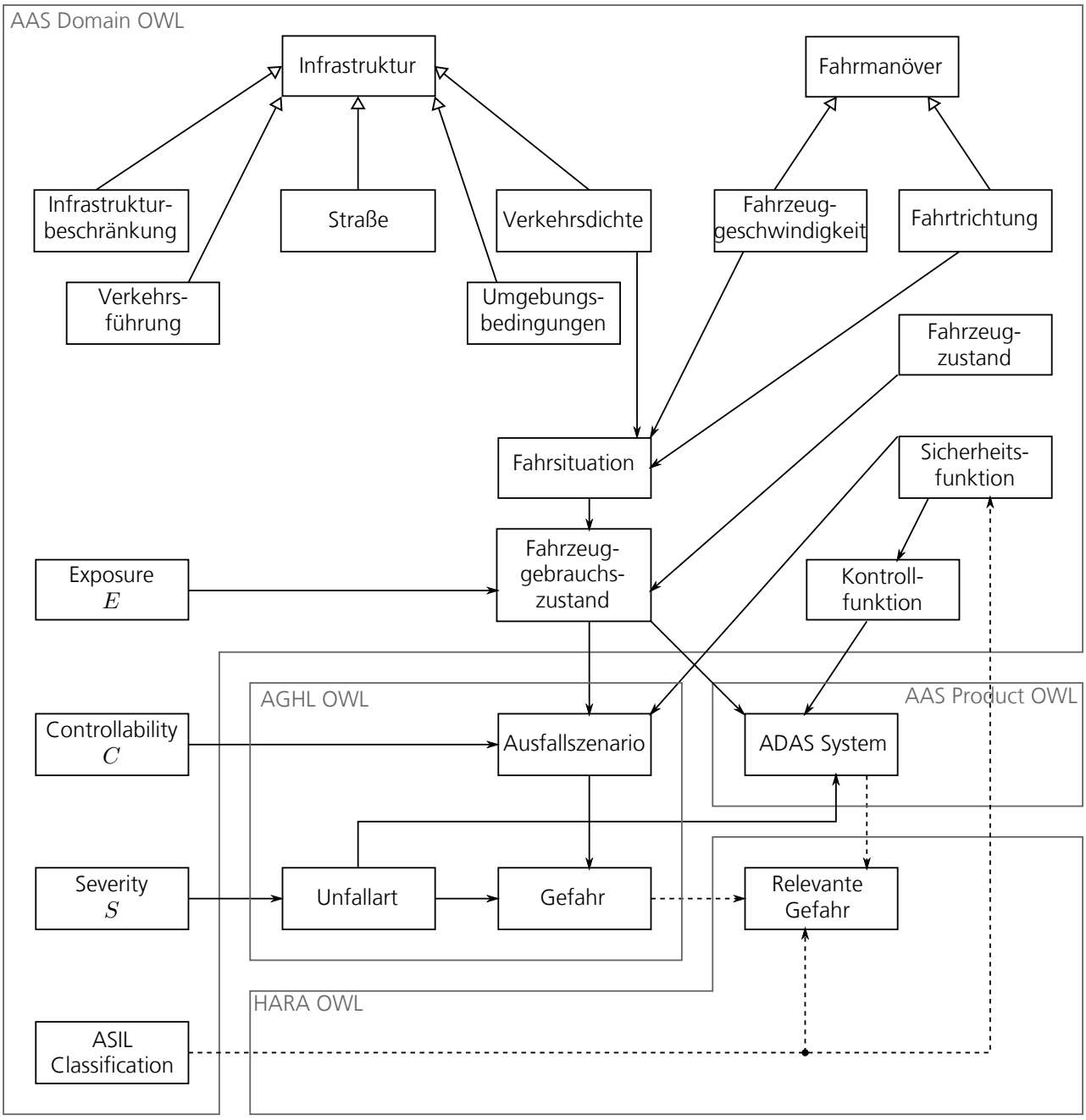

Abbildung 8-3: Hier wird ersichtlich, wie die automatisierte Risikoanalyse auf die in Kapitel 5 beschriebenen basalen Ontologien aufsetzt.

Insbesondere SPARQL als Abfragesprache ermöglicht Operationen auf der OWL bzw. dem unterliegenden RDF, die denen auf konventioneller Datenbanktechnologie nahe kommen. Somit kann hier konzeptionell weiter sauber gearbeitet werden. Wenn jedoch technische Grenzen (wie etwa Ausführungsgeschwindigkeit) oder die in Kapitel 6 erwähnten konzeptionellen Grenzen der OWL Beschreibungslogik für eine Inferenz erreicht werden, so konnten im vorliegenden Fall durch den Einsatz von SPARQL Migrationslösungen definiert werden. Somit wird zwar in einzelnen Fällen keine volle Inferenz mehr durchgeführt, jedoch können äquivalente logische Aussagen mittels SPARQL Abfragen definiert werden. Diese Transformation per Abfrage anstelle von Reasoning ist technisch deutlich effizienter.

Konkret dauert eine Inferenz, die die formalisierte Edition der Norm verwendet, mehrere Stunden. Eine vom Resultat äquivalente Abfrage mittels SPARQL kann in wenigen Sekunden durchgeführt werden.

8.3 Instanziierung am Beispiel ACC / EBS

Im Weiteren wird die gerade beschriebene prototypische Werkzeugkette auf das Beispiel des Notbremsassistenten EBS angewendet werden, dazu ist eine angepasste und verfeinerte Übersicht in Abbildung 8-4 zu sehen. Dabei sind die Schritte (1) bis (6) dargestellt, welche in der weiteren Beschreibung weiter verfeinert werden.

Systemgrenzen und Domänenattribute (1)

Der neue Notbremsassistent EBS soll insbesondere gefährliche Situationen außerhalb geschlossener Ortschaften adressieren. Dieses kann mit existierender ACC Sensorik umgesetzt werden. Neue Anwendungen, wie dieser Notbremsassistent, können so ohne hohe Zusatzkosten durch zusätzliche Sensorik umgesetzt werden. Schließlich verfügt das Fahrzeug bereits über ein ESP (elektronisches Bremsen). So sollen die zusätzlichen Kosten für das neue AAS möglichst gering gehalten werden, da für solche Systeme eine geringe Zahlungsbereitschaft vorliegt [Zwe06]. Gleichzeitig legte eine Studie von Unfallzahlen nahe, dass durch die Einführung des EBS insbesondere die Anzahl von Unfällen reduziert werden könnte, da hier zwei wichtige Unfallursachen [Sta10] (zu geringer Abstand und überhöhte Geschwindigkeit) adressiert werden können.

Dabei ergibt sich nun eine Reihe von Domänenattributen, die dem neuen System annotiert werden können. Betrachtet wird dabei das Beispiel in Abbildung 8-4:

- *Umgebungsbedingungen*: Das System muss in allen Wetterlagen funktionieren, dazu zählt eine regnerische Nacht genauso wie ein sonniger, trockener Sommertag. Gleichzeitig soll das System nur außerhalb geschlossener Ortschaften verwendet werden können, also Fahrten auf Landstraßen und Autobahnen. Gleichzeitig wird so der Geschwindigkeitsbereich vorgegeben, das Notbremssystem soll erst ab 65 km/h aktiv sein. Im Weiteren wird eine Situation betrachtet, in der für das eigene Fahrzeug freie Fahrt besteht, jedoch Folgeverkehr vorhanden ist.

- *Sicherheitsfunktion*: Über den Notbremsassistenten wird über das ESP gegebenenfalls eine Gefahrenbremsung, also eine Vollbremsung, ausgelöst. Dabei soll das System nur dann aktiv sein, wenn sich das Fahrzeug auf (nahezu) gerader Strecke befindet. Somit reduziert sich wiederum die Anzahl der möglichen gefährlichen Einsatzsituationen. Weiterhin kann so die aktuelle statische Sensorkonfiguration genutzt werden, die ein Radar mit vergleichsweise kleinem Öffnungswinkel verwendet. Somit greift dieses System explizit nur in die longitudinale Dynamik des Fahrzeugs ein.

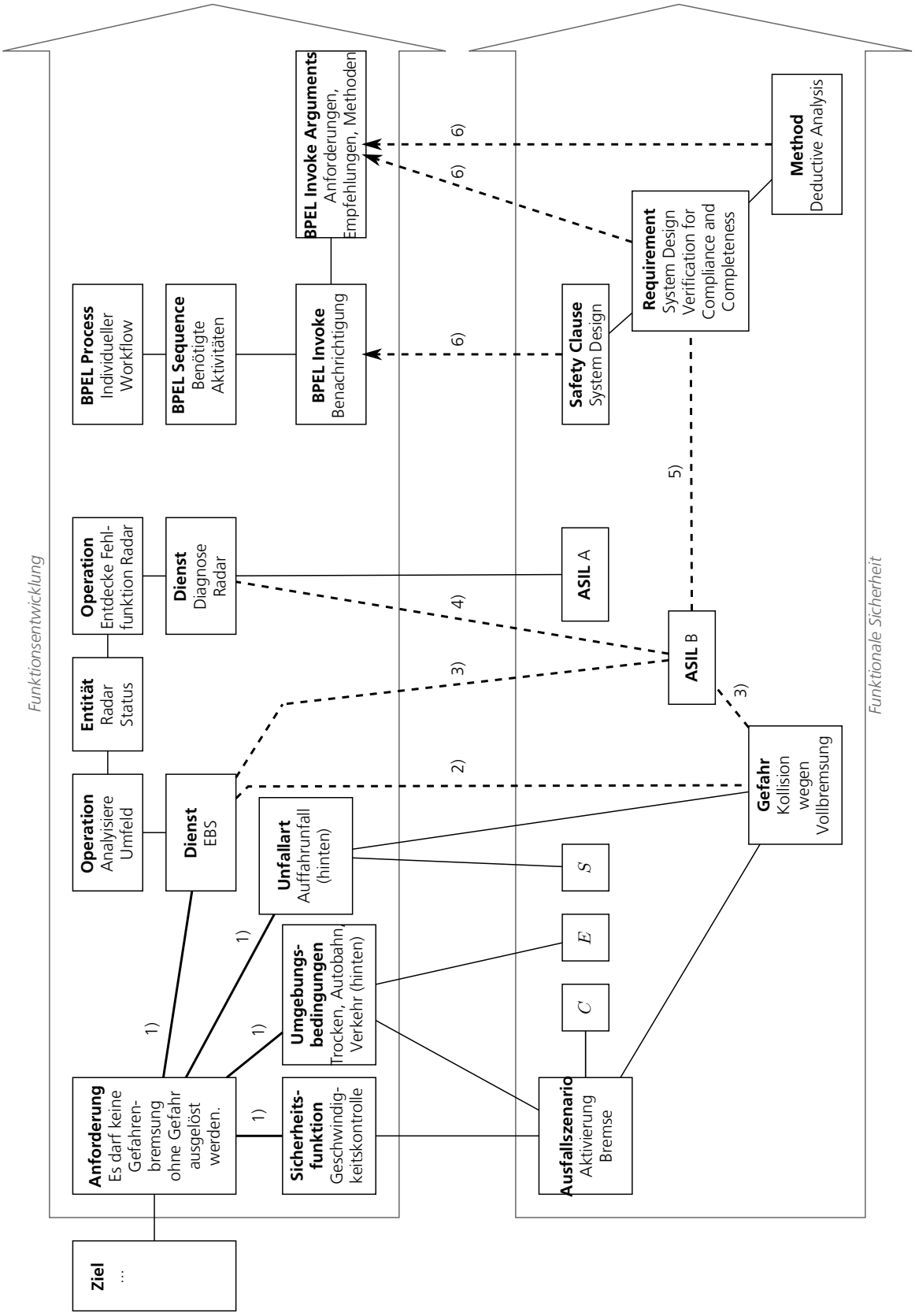

Abbildung 8-4: Konkretisiertes Beispiel, die Schritte (1) bis (6) werden detailliert erläutert. Inferierte bzw. konstruierte Assoziationen sind gestrichelt dargestellt.

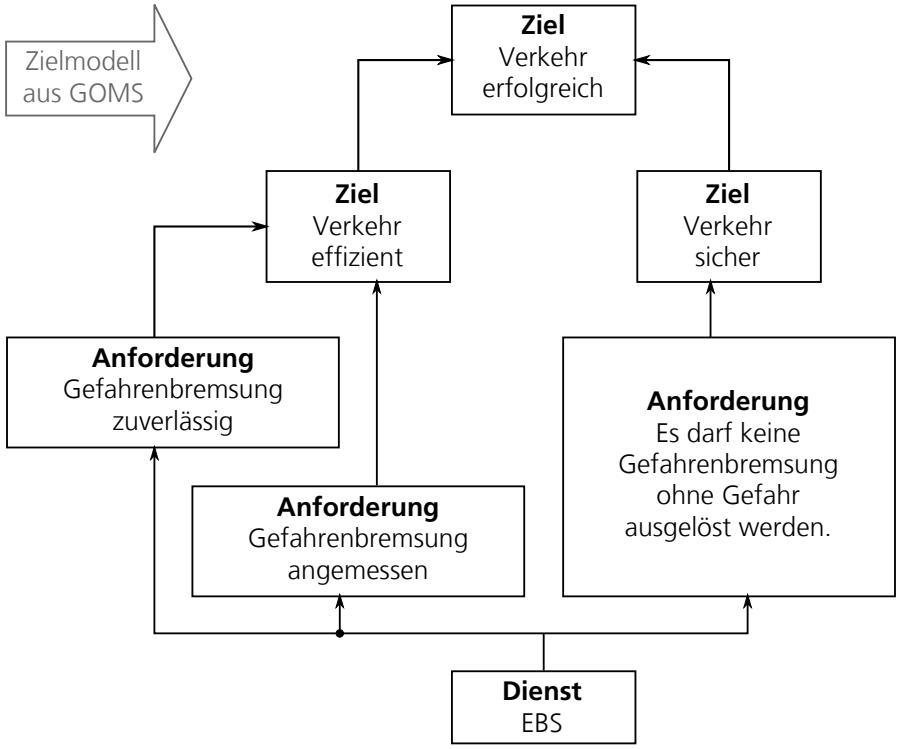

Abbildung 8-5: Visualisierung des Zielmodells.

- *Unfallarten*: Das System greift longitudinal in die Dynamik des Fahrzeugs ein. Daher kommen insbesondere Kollisionen mit Fahrzeugen in derselben Spur infrage. Aufgrund der vorliegenden Verkehrssituation (nur Folgeverkehr) ist vor allem eine Fehlauslösung des Systems gefährlich. Wenn der Notbremsassistent bei freier Strecke aktiv wird und ein folgendes Fahrzeug existiert, wird dessen Fahrer von dieser Situation vollkommen überrascht.

Anforderungen (1)

Während des *Requirements Engineering* wurden Anforderungen erhoben, dabei wurde KAOS verwendet. So lassen sich generische Ziele formulieren, die sowohl für einen menschlichen Operator wie für ein technisches System gelten. Exemplarisch können für das EBS bzw. einen Autofahrer die verkehrlichen Ziele Verkehrssicherheit und Verkehrseffizienz formuliert werden. Für die Systemgrenzen des EBS leiten sich drei wesentliche funktionale Anforderungen ab.

1. Eine Notbremsung muss dann erfolgen, wenn mit hoher Zuverlässigkeit ein Auffahrunfall droht. Dazu gehört eine gesicherte Erkennung des anderen Verkehrsteilnehmers wie auch der Prädiktion der Fahrdynamik.
2. Eine Notbremsung darf nicht erfolgen, wenn keine ausreichende Wahrscheinlichkeit für einen bevorstehenden Unfall besteht.
3. Eine Notbremsung darf nicht erfolgen, wenn der Fahrer durch vorausschauende Fahrweise den Unfall selber verhindern kann.

Diese Ziele gelten sowohl für das technische System als auch für das Verhalten des Autofahrers. Die Zielstruktur kann also sowohl in einem GOMS Modell, wie auch einem KAOS Modell genutzt und weiter verfeinert werden. Dabei wird die angegebene Zielstruktur nicht mehr vom Autofahrer *erwartet* (Erwartung), sondern nun vom technischen System *gefordert* (Anforderung). So können zudem die Ziele wiederverwendet werden. Das Zielmodell ist in Abbildung 8-5 visualisiert (ein Bildschirmfoto findet sich in Abbildung C-3 im Anhang). Auch mit Hinsicht auf die GSN kann die

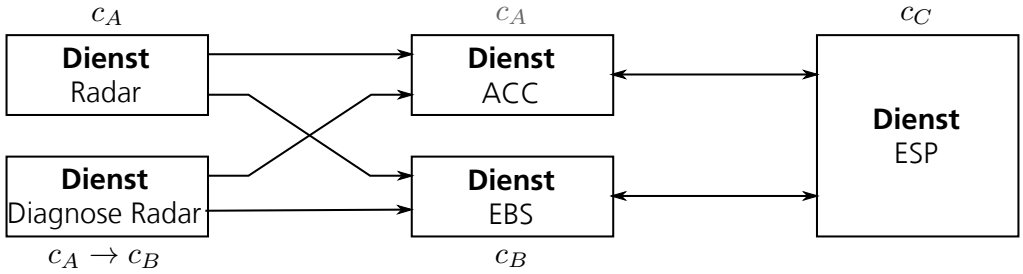

Abbildung 8-6: Das Notbremssystem EBS nutzt den Radarsensor mit Diagnose, der durch ein ACC bereits vorhanden ist. Über das ESP wird eine Verzögerung ausgelöst.

gegebene Zielstruktur verwendet werden, um die Sicherheitsargumentation zu führen. Konkret können nun für EBS die Ziele weiter verfeinert werden. Dabei haben Studien ergeben, dass die *Time To Collision* (TTC) t_{TTC} sich als Regelgröße für einen Notbremsassistenten eignet.

1. Eine Notbremsung sollte dann und nur dann erfolgen, wenn

$$(t_{\mathrm{TTC}} < T_{\mathrm{kritisch}}) \wedge (q_{\mathrm{TTC}} > Q_{\mathrm{kritisch}}) \tag{8.1}$$

   gilt. Dabei muss t_{TTC} einen kritischen Wert T_{kritisch} unterschreiten, gleichzeitig muss für die Qualität q_{TTC} des TTC Wertes ein minimales Qualitätsniveau Q_{kritisch} gegeben sein. Somit deckt diese Funktion die oben genannten Ziele ab.

2. Weiterhin gilt es hier, die Systemgrenzen

$$(v_x > v_{\mathrm{min}}) \wedge (\delta_{\mathrm{L}} < \delta_{\mathrm{L,max}}) \wedge (\dot{\psi} < \dot{\psi}_{\mathrm{max}}) \tag{8.2}$$

   einzuhalten. So soll eine Aktivierung erst dann erfolgen können, wenn die Fahrzeuglängsgeschwindigkeit v_x die minimale Geschwindigkeit v_{min} für die Aktivierung übersteigt. Weiterhin darf der Lenkwinkel δ_{L} nicht größer als ein Maximalwert $\delta_{\mathrm{L,max}}$ sein. Dasselbe gilt für die aktuelle Gierrate $\dot{\psi}$ (Beschränkung $\dot{\psi}_{\mathrm{max}}$).

3. Das System soll zeitdiskret realisiert werden, dabei soll eine Abtastrate von 20 Hz garantiert werden.

4. Weitere Anforderungen ergeben sich aus der Einstufung als sicherheitskritische Funktion, was insbesondere die Ausführung der Entwicklung betrifft.

5. Weitere Anforderungen ergeben sich daraus, dass das System in seiner geplanten Konfiguration auf eine entsprechende Kundenakzeptanz treffen soll.

Neben den genannten Anforderungen kommen für eine Serienumsetzung weitere Anforderungen hinzu, auf die hier nicht näher eingegangen wird, um eine angemessene Komplexität des vorliegenden Beispiels zu behalten.

Systemarchitektur (1)

Für die Systemarchitektur bedeutet die Einführung des EBS einen parallelen Zugriff auf den Bremseingriff des ESP, was in Abbildung 8-6 zu erkennen ist. ACC und EBS verwenden beide den Radar mit seiner Diagnose. Dabei hat die Einführung des EBS Rückwirkungen auf die Diagnose des Radars, da das EBS eine sehr hohe ASIL Einstufung erfährt.

Vorläufige Risikoanalyse (2, 3)

Nach der Definition von funktionalen Anforderungen und der abgeleiteten Systemarchitektur kann untersucht werden, wie sich die Modifikation des existierenden Systems insbesondere

unter dem Gesichtspunkt der funktionalen Sicherheit auswirkt. Dabei ist die Durchführung einer vorläufigen Risikoanalyse notwendig. In diesem Fall ergibt sich für die Anforderung

„Es darf keine Gefahrenbremsung ohne Gefahr ausgelöst werden."

über die annotierten Domänenattribute eine Klassifikation nach ASIL B.

Einflussanalyse unter Verwendung der OWL und eines Reasoners (4)

Eine besonders wichtige Fragestellung ergibt sich daraus, was für Konsequenzen die neue Systemarchitektur auf die Entwicklung hat. Und dabei insbesondere, was für Konsequenzen sich durch das neue EBS und seine ASIL Klassifikation für das bereits entwickelte Basissystem mit ACC ergeben. Im konkreten Fall sind die Abhängigkeiten in Abbildung 8-6 zu sehen. In diesem Fall ergibt sich, dass auch die Plausibilisierung des Sensorsignals (Diagnose) nach ASIL B entwickelt werden muss (siehe Bildschirmfoto in Abbildung C-4 im Anhang).

Bewertung und Instanziierung einer Entwurfsalternative (5, 6)

Nachdem der ASIL für alle betroffenen Komponenten bestimmt wurde, kann nun für die jeweilige Klassifikation des ASILs und der Komponente ein Prozess konform zur ISO DIS 26262 inferiert werden. Dieses wird dadurch möglich, dass mit einem ASIL bestimmte *requirements and recommendations* assoziiert sind, die wiederum den Einsatz spezifischer *methods* erfordern. Weiterhin sind diese mit bestimmten *safety clauses* (Phasen) im Entwicklungsprozess verknüpft, sodass ein spezifischer Entwicklungsprozess zugeschnitten (Tailoring) werden kann. Abschließend kann das Modell, wie in Kapitel 5 und Kapitel 6 beschrieben, per NuSMV (Abbildung C-5) verifiziert und per BPEL instanziiert werden (siehe Abbildung C-6). Darüber hinaus kann HTML Dokumentation inklusive Visualisierung der Prozessabläufe generiert werden (siehe Abbildung C-7).

8.4 Implementierung ACC / EBS

Weiterhin kann der neue Notbremsassistent EBS als Dienst mittels OWL-S, VSDL und VSD beschrieben und über VPEL orchestriert werden. Etwas vereinfacht sieht der VPEL Ablauf folgendermaßen aus (siehe auch Abbildung 8-7, bzw. die Bildschirmfotos in den Abbildungen C-8, C-9 und C-10 im Anhang):

1. *onWait* 50 ms: Alle 50 ms (entspricht 20 Hz) wird der vorliegende VPEL Prozess ausgelöst.
2. *invoke* getEgoState() / getTTC() / getPsi(): Die gesuchten Messgrößen werden synchron mittels einer Serviceschnittstelle angefordert.
3. *invoke* analyzeEnvironment(): Aus den betrachteten Messwerten wird entschieden, ob eine Gefahrenbremsung eingeleitet werden muss. Dabei wird die in den Anforderungen beschriebene Logik verwendet, welche vor allem auf TTC Messungen basiert.
4. *invoke* triggerBrake(): Liegen alle notwendigen Voraussetzungen für eine Gefahrenbremsung voraus, so wird über das ESP die Notbremsung eingeleitet.

Im Fall des vorliegenden Systems kann insbesondere die als Anforderung spezifizierte Logik direkt als Spezifikation zur technischen Umsetzung herangezogen werden.

An dieser Stelle soll auf eine Besonderheit der hier vorliegenden Spezifikation hingewiesen sein, die auf der Basis der OWL beschrieben wurde. Die OWL eignet sich ab Version 2 dazu, konkrete logische Aussagen zu treffen, die auf quantitativen Werten beruhen (z.B. dass ab einem

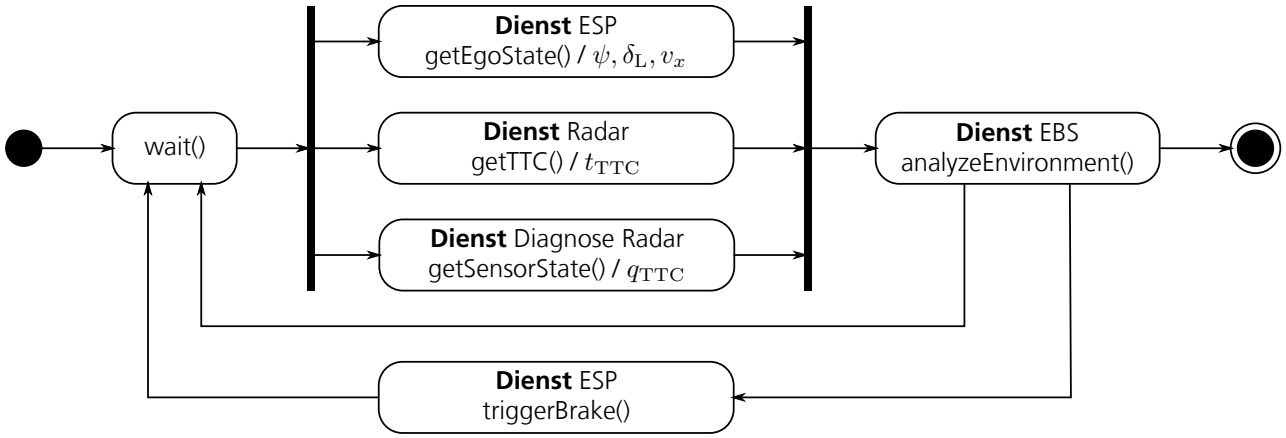

Abbildung 8-7: Orchestrierung des EBS Dienstes (vereinfacht).

bestimmten Wert t_{TTC} eine Bremsung eingeleitet werden soll). Weiterhin ist dabei immer noch eine Sprache vorliegend, welche über eine geschlossene mathematische Theorie (insbesondere eine modelltheoretische Semantik) verfügt. Gleichzeitig verfügt die OWL insbesondere durch ihre XML basierte Serialisierung über einfache Interoperabilität und Austausch von logischen Modellen.

Evolution und Individualisierung

Die Nutzung der OWL macht in diesem Fall eine vergleichsweise einfache *Evolution* oder *Individualisierung* der vorliegenden Logik möglich. Dieses funktioniert, indem eine OWL basierte Spezifikation als Basis für eine individualisierte Logik genutzt werden kann. Liegt solch eine individualisierte Logik vor, kann diese mittels eines Reasoners auf Konsistenz geprüft werden. So kann geprüft werden, ob ein spezialisiertes System Widersprüche in Bezug auf seine generische Basis aufweist. Liegt ein solcher Widerspruch vor, kann dieser *bewiesen* werden. Es sind also keine Tests notwendig, um einen ganzen Zustandsraum abzuprüfen.

Die oben beschriebene Spezifikation kann mittels Beschreibungslogik in eine Klassifikationsaufgabe überführt werden. Demnach kann eine gefährliche Situation mit den notwendigen Bedingungen

$$\begin{aligned} C_{\mathrm{Danger}} &\sqsubseteq C_{\mathrm{Situation}} \\ C_{\mathrm{Danger}} &\sqsubseteq \geq 1\ R_{\mathrm{hasTTC}} \cdot T_{\mathrm{kritisch}} \\ C_{\mathrm{Danger}} &\sqsubseteq \geq 1\ R_{\mathrm{hasQuality}} \cdot Q_{\mathrm{kritisch}} \end{aligned} \tag{8.3}$$

formuliert werden. Dabei kann nun eine mögliche Spezialisierung geprüft werden. Treten dort Widersprüche durch Fehler in einer Spezialisierung

$$\begin{aligned} C_{\mathrm{DangerSpecial}} &\sqsubseteq C_{\mathrm{Danger}} \\ C_{\mathrm{DangerSpecial}} &\sqsubseteq < 1\ R_{\mathrm{hasQuality}} \cdot Q_{\mathrm{kritisch}} \end{aligned} \tag{8.4}$$

auf, können diese durch den Reasoner erkannt werden. In diesem Fall wird eine solche Klasse als

$$C_{\mathrm{DangerSpecial}} \sqsubseteq \bot \tag{8.5}$$

klassifiziert, was einem Widerspruch in der Terminologie der Beschreibungslogik entspricht. Damit deutet diese beispielhafte Implementierung an, wie trotz (oder gerade wegen) des Einsatzes von Technologien aus der Domäne Internet eine flexible und sichere Evolution und Individualisierung von Funktionen auch im Automobil erzielt werden können.

8.5 Zusammenfassung des Kapitels

Als Anwendungsbeispiel wurde die in Abschnitt 4 beschriebene Anwendung eines ACC mit Notbremsassistenz weiter verfeinert.

Es wurde eine Werkzeugkette prototypisch implementiert und praktisch im Rahmen eines Demonstrators angewendet. Diese basiert ausnahmslos auf offenen Standards (OWL, SWRL, SPARQL, XSLT), die mit ebenfalls offenen und generischen Werkzeugen (Eclipse, Apache Ant, Protégé, OWL API, HermiT, Graphviz) umgesetzt wurden. Mit Objectiver wurde weiterhin ein kommerzielles Werkzeug angebunden. Diese Werkzeugkette umfasst dabei insbesondere die Verknüpfung von Domänenwissen aus den Bereichen der *Funktionsentwicklung* von Assistenz- und Automationssystemen sowie der *Funktionalen Sicherheit*. Somit können Anforderungen aus einer fremden Domäne für das eigene Handeln abgeleitet werden. Durch die Verwendung der offenen und generischen Basiswerkzeuge (insbesondere Eclipse, Apache Ant, Graphviz) liegt eine gute Erweiterbarkeit vor. Diese könnte für eine Verknüpfung mit Methoden und Maßnahmen aus dem Bereich *Faktor Mensch* genutzt werden.

Die Implementierung ist serviceorientiert beschrieben, wobei hier die Implementierung des Dienstes an sich sogar in OWL erfolgt ist. Das soll andeuten, wie durch die Verwendung der OWL als Internettechnologie eine flexible und sichere Evolution und Individualisierung logischer (nicht algorithmischer) Dienste erfolgen kann. Dabei wird die formale Basis der OWL genutzt, indem eine Implementierung gegen eine Referenzspezifikation geprüft wird. Somit kann bezüglich der Konsistenz ein mathematischer Beweis erfolgen, welcher einen funktionalen Test ersetzt.

8.6 Zusammenfassung des Teils: Implementierungen und praktische Anwendung

An dieser Stelle schließt der dritte Teil dieser Arbeit, der sich den Softwareimplementierungen von AAS sowie einer beispielhaften praktischen Anwendung widmet. Einen Gesamtüberblick über die Struktur dieser Arbeit zeigt Abbildung 8-8.

Dabei baut das Konzept der Softwareimplementierungen (Kapitel 7) auf den formalisierten Modellen im Rahmen des Entwicklungsprozesses auf. Offene Standards werden adaptiert, um eine bessere Eignung der AAS Domäne zu ermöglichen. Nichtsdestotrotz sind die definierten Sprachen in ihrer möglichen Anwendung weitestgehend domänenunabhängig. Weiterhin wurde ein Codegenerator prototypisch implementiert.

Anhand der Erweiterung eines ACC Systems zu einem Notbremssystem wurde eine solche Entwicklung über verschiedene Entwicklungsstränge beispielhaft durchgeführt (Kapitel 8). Dazu wurde eine prototypische Werkzeugkette entwickelt, die weiterhin aufwändige Modelle der ISO DIS 26262 zusammen mit einer automatisierten Risikoanalyse nutzt [Jos10].

Dabei wurde ersichtlich, dass der Modellierungsaufwand zwar hoch ist, es sich jedoch fast ausschließlich um anwendungsunabhängige Modelle handelt. Wesentliche Basismodelle (wie etwa das Vorgehensmodell) sind sogar domänenübergreifend nutzbar. Somit haben die Modelle einen sehr hohen Anteil an Wiederverwendbarkeit, beispielsweise das fast 1000 Anforderungen umfassende Modell der ISO DIS 26262.

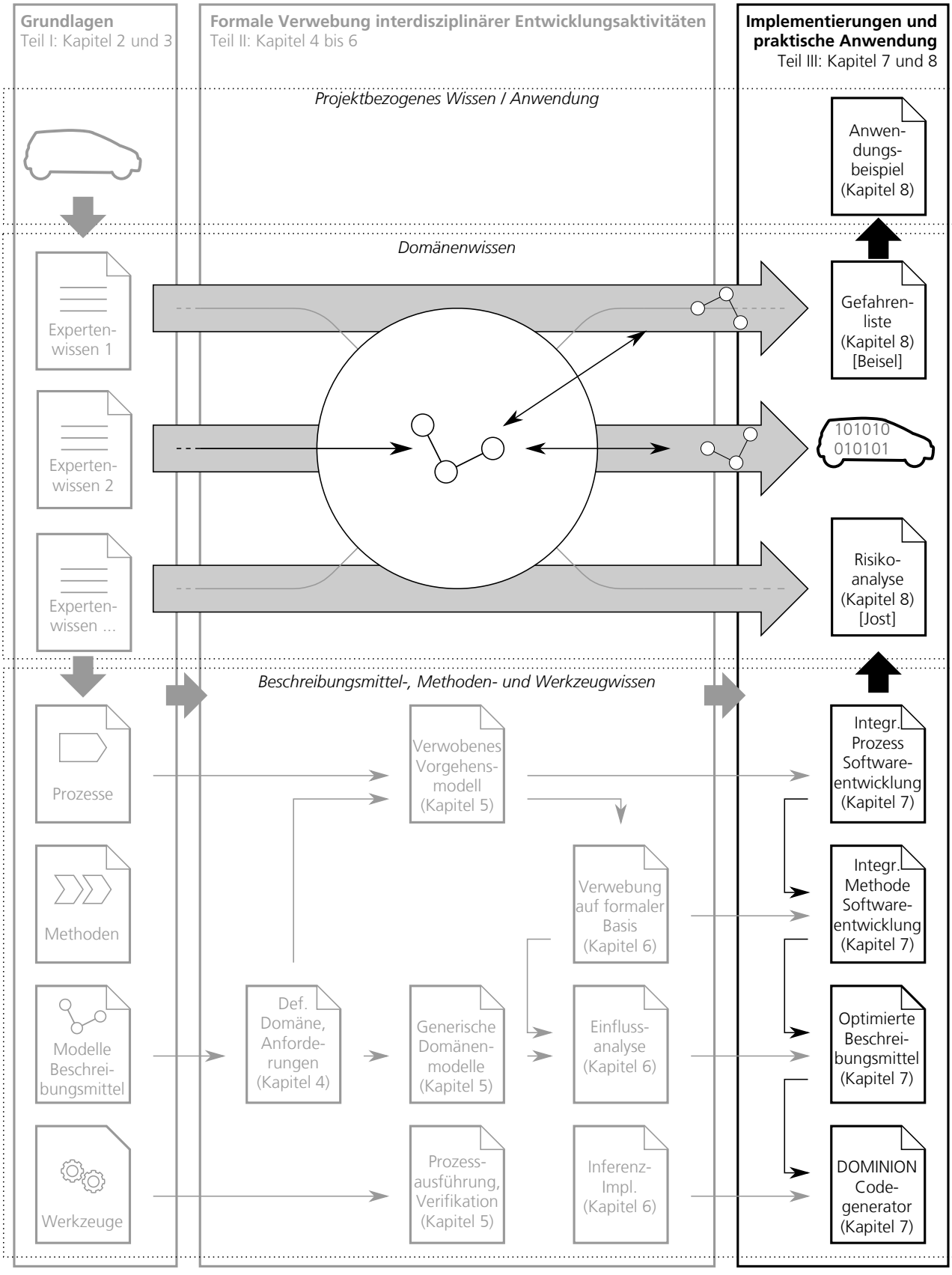

Abbildung 8-8: Struktur der Arbeit.

Teil IV

Bewertung

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

Die vorliegende Arbeit leistet einen Beitrag zur Beantwortung der Forschungsfragen *Formalisierung des Entwicklungsprozesses* und *Verwebung interdisziplinärer Entwicklungsaktivitäten*.

Zunächst wurden die Grundlagen in Bezug auf Vorgehensmodelle (Kapitel 2) sowie Domänenwissen und Anforderungsmodellierung (Kapitel 3) eingeführt. In Kapitel 2 wurden die wichtigen Referenzprozesse der ISO DIS 26262 und RESPONSE 3 CoP der AAS Domäne strukturiert und analysiert. Verbesserungspotential wurde dabei insbesondere durch eine Formalisierung des Entwicklungsprozesses identifiziert. So können etwa Inkonsistenzen vermieden werden, die aktuell noch in den genannten Referenzprozessen vorliegen.

Weiterhin wurden allgemeine Vorgehensmodelle beschrieben und im Kontext der AAS Entwicklung bewertet. Für den Einsatz bei der Serienentwicklung sicherheitskritischer AAS eignen sich insbesondere Vorgehensrahmenwerke, da diese ein Tailoring eines Entwicklungsprozesses ermöglichen. Darüber hinaus wurden die Phasen des *Requirements Engineering* und *Domain Engineering* aus der Literatur eingeführt, die eine zentrale Rolle in Bezug zur Verwebung interdisziplinärer Entwicklungsaktivitäten spielen. In Kapitel 3 wurden Beschreibungsmittel, Methoden und Werkzeuge vorgestellt, deren Nutzung sich im Rahmen der genannten Phasen des Requirements und Domain Engineering eignet. Insbesondere vor dem Hintergrund einer offenen und flexiblen Formalisierung eignet sich der Einsatz der Ontologiemodellierungssprache OWL. Diese ermöglicht eine wissensbasierte Modellierung einer Anwendungsdomäne im Rahmen des Domain Engineering, was dann im Requirements Engineering aufgegriffen werden kann. Für das Requirements Engineering im Kontext der Verwebung interdisziplinärer Entwicklungsaktivitäten eignen sich zielorientierte Methoden. Dabei liegen mit KAOS (*Funktionsentwicklung*), GOMS (*Faktor Mensch*) und der GSN (*Funktionale Sicherheit*) drei Methoden aus den Teildomänen der AAS Entwicklung vor, die sich über ein gemeinsames Zielmodell verweben lassen.

Die drei genannten Teildomänen wurden in Kapitel 4 aufgegriffen und als interdisziplinäre *Entwicklungsstränge* gekapselt. Dazu wurde eine Definition und Strukturierung der Begriffe *Multidisziplinarität*, *Transdisziplinarität* und *Interdisziplinarität* gegeben. Insbesondere die Interdisziplinarität ist für die vorliegende Arbeit von zentraler Bedeutung. Dabei liegt eine lose gekoppelte Entwicklung vor, die Wissensaustausch ohne Methodenaustausch vornimmt. Weiterhin wurde ein Anwendungsbeispiel skizziert. Dieses umfasst die Entwicklung eines ACC Systems, welches um einen Notbremsassistenten erweitert werden soll. Das Beispiel und die Begriffsdefinition wurden genutzt, um konkretisierte Anforderungen an eine *Formalisierung des Entwicklungsprozesses* und *Verwebung interdisziplinärer Entwicklungsaktivitäten* abzuleiten.

Insbesondere zur *Formalisierung des Entwicklungsprozesses* wurde in Kapitel 5 ein Vorgehensmodell definiert, welches auf formaler Basis verschiedene Entwicklungsstränge verwebt. Dabei wird im Domain Engineering das Domänenwissen der Entwicklungsstränge in OWL Ontologien gekapselt, welches dann im Requirements Engineering aufgegriffen wird. Das Vorgehensmodell ist rekursiv, so dass Entwicklungsstränge weitere Teilentwicklungsstränge umfassen. Die abstrakte Syntax des Vorgehensmodells wurde in OWL formalisiert und steht als Ontologie zur Verfügung. Für die Ablauflogik wurde eine temporale Semantik über Kripke Strukturen ausgedrückt, deren Eigenschaften durch LTL Formeln automatisch verifiziert werden können. Eine prototypische Werkzeugunterstützung wurde umgesetzt. Dazu wird eine Instanz

des Vorgehensmodells per XSLT in einen ausführbaren Workflow (BPEL), ein Verifikationsmodell (NuSMV) sowie Dokumentation (HTML) überführt.

Kapitel 6 nutzt und verfeinert insbesondere die im Rahmen des Vorgehens modellierten Ontologien zur *Verwebung interdisziplinärer Entwicklungsaktivitäten*. Durch eine Verknüpfung des Domänenwissens über Axiome und Ontologien kann ein Wissenstransfer im Sinne der Definition von Interdisziplinarität erfolgen. Dabei findet ein Wissenstransfer, jedoch kein Methodenaustausch statt. Dieses wurde anhand der zielorientierten Requirements Engineering Methoden GOMS und KAOS verdeutlicht. Weiterhin wurde eine Einflussanalyse formalisiert, die auf der formalen Basis der OWL aufsetzt und die Inferenz nutzt, um implizite Anforderungen zu entdecken. Diese Modellverknüpfung und Inferenz dient dabei vor allem dem Zweck, Entwurfsalternativen gegeneinander abzuwägen, indem der Einfluss eines einzelnen Entwicklungsstranges auf den gesamten Entwicklungsprozess untersucht wird. Dadurch wird der Entwurf besser und genauer planbar. Zur technischen Umsetzung wurden generische Komponenten entwickelt, die eine Inferenz bzw. einen Wissenstransfer nach den Standards OWL (Beschreibungslogik), SWRL (Regeln) und SPARQL (Abfragen) ermöglichen. Diese Komponenten sind im Rahmen einer Apache Ant basierten Werkzeugkette einsetzbar. Als Reasoner ist HermiT eingebunden, RDF wird durch des Jena Framework unterstützt.

In Kapitel 7 wurde ein möglicher Weg aufgezeigt, die generischen und abstrakten konzeptuellen Modelle aus den Ontologien in der (softwaretechnischen) *Funktionsentwicklung* aufzugreifen und an Implementierungen zu binden. Dazu wurden modifizierte Beschreibungsmittel vorgeschlagen, die Internetstandards im Bereich *Service-orientierter Architekturen* aufgreifen und auf ihre Nutzung im Kontext sicherheitskritischer Systeme anpassen: die in-Vehicle Service Languages VSDL, VSD und VPEL. Dabei wird insbesondere eine Codegenerierung als Baustein eines modellgetriebenen Ansatzes verwendet. Diese ersetzt nach Bedarf die Flexibilität zum Entwicklungszeitpunkt durch eine technisch effiziente und statische Bindung. Diese Lösung kann weiterhin als Migrationsstrategie dienen, bis die Implementierungen regulärer Internettechnologien über ausreichende weiche Echtzeitfähigkeiten verfügen. Die Nutzung serviceorientierter Technologien ist der logische Schritt in der Fortschreibung des Technologietransfers von Internettechnologie in den Bereich der eingebetteten Systeme. Ein XSLT Prototyp eines Codegenerators wurde dabei umgesetzt, der die in-Vehicle Service Languages auf die DOMINION Laufzeitumgebung abbildet.

Das Kapitel 8 greift das Anwendungsbeispiel (ACC plus Notbremsassistent) auf. Anhand der in den vorangegangenen Kapiteln beschriebenen Methoden wurde eine exemplarische Umsetzung gezeigt. Dabei wurde vor allem auf die Verwebung der *Funktionsentwicklung* mit der *Funktionalen Sicherheit* eingegangen. Die Änderung der ACC Architektur durch Einführung des Notbremsassistenten hat Einfluss auf das Gesamtsystem. Dazu gehört insbesondere die Durchführung von Sicherheitsmaßnahmen nach ISO DIS 26262. Durch die Verwebung der beiden Entwicklungsstränge kann für die neuen und geänderten Komponenten ein spezieller, individueller Workflow logisch gefolgert werden. Dieser umfasst die Menge an Maßnahmen, die zur Durchführung der Entwicklung nach dem Sicherheitsstandard notwendig sind. Weiterhin wurde die Implementierung des Notbremsassistenten skizziert, die durch Verwendung formaler Logik (OWL) einen möglichen Lösungsweg für *Individualisierung und Evolution* im Rahmen sicherheitskritischer kooperativer AAS aufzeigt.

9.2 Verwandte Arbeiten

Im Bereich der Verwebung interdisziplinärer Entwurfsaktivitäten existieren eine Reihe verwandter Ansätze, in die sich diese Arbeit einbettet bzw. die sich gegenseitig ergänzen.

Dazu gehört etwa *Compliance Flow* [CCM08], welches die Konformität eines Unternehmensprozesses mit einem Referenzprozess untersucht. Dazu werden einzelne Entwurfsartefakte mittels einer Ontologie kategorisiert und so vergleichbar gemacht. Es ist jedoch nicht ersichtlich, auf welchem Beschreibungsmittel die Ontologie fußt, und die Analyse ist nicht formalisiert beschrieben.

Die Arbeit *Zur Formalisierung technischer Normen* [Hän08] behandelt Methoden zur Formalisierung von Normen. Dabei werden Ontologien als Beschreibungsmittel von Konzepten eines Standards verwendet, mit Petrinetzen werden Abläufe modelliert. Durch die explizite Modellierung von Abläufen mit Petrinetzen ist jedoch keine Verwebung mit anderen Modellen möglich, also Verknüpfung von Methoden und Anforderungen, die eine zielgerichtete Unterstützung ermöglichen würden.

Formalisierte Terminologien technischer Systeme und ihrer Zuverlässigkeit [Sch09b] beschreibt eine Methode zur Formalisierung von Begriffen. Dazu wird ein metasprachliches Modell definiert und ein Terminologiemanagementsystem entwickelt. Dessen Abbildung auf die OWL ist für die Zukunft geplant.

Bei *Integration of Safety and model-driven design* [dMBSA08] werden UML Modelle um Attribute für eine Sicherheitsanalyse erweitert. Somit können aus UML Softwaremodellen *Failure mode, effects, and criticality analysis* (FMECA) und *Fault Tree Analysis* (FTA) Bäume erzeugt werden, die dann von einem Sicherheitsexperten ausgewertet werden. Dieser Ansatz beleuchtet eine konkrete Modellverknüpfung, beschreibt jedoch noch keine automatisierte Analyse, wie sie etwa durch Verwendung der OCL [OMG10] möglich wäre.

Eine ausführliche Ausarbeitung zur Standardkonformität findet sich bei *Managing Standards Compliance* [EFA+99], was die Modellierung und formale Analyse von Konformität umfasst. Der Ansatz steht jedoch für sich, eine Verknüpfung mit anderen Domänen, etwa der Softwareentwicklung, ist nicht im Detail erläutert.

Eine Verknüpfung von Entwurf und Test liefert *Model based development in Automotive* [MHGK03]. Unter anderem wurden hier die Werkzeuge ASCET und ARTISAN miteinander verknüpft. Während der Modelltransformation findet weiterhin ein Prüfung von Entwurfsregeln statt.

Technische Basis dieser Arbeit bilden Standards des W3C, auch wenn die methodischen Aspekte sich weitestgehend auch mit Standards der OMG (UML zur Modellierung, die *Object Constraint Language* (OCL) [OMG10] zur Analyse, *Query View Transformation* (QVT) [OMG08a] und *Model to Text* (M2T) [OMG08b] zur Transformation) nachbilden ließen. Die Entscheidung insbesondere für die OWL ist ihre Offenheit und der kompakte, formale und konzeptionell saubere Sprachumfang.

Besondere Erwähnung verdient der Ansatz *Reasoning on Domain Knowledge and Technical Standards to Support the Development of Safety-Critical Automotive Systems* [Jos], der mit dieser Arbeit eng verzahnt ist und insbesondere im Rahmen des Projektes DeSCAS entstand. Der Ansatz beschreibt die Modellierung von Domänenontologien unter Verwendung (technischer) Standards und Normen. Dabei entstand eine umfassende Ontologie der ISO DIS 26262, die im Rahmen der vorliegenden Arbeit zur Demonstration der *Formalisierung des Entwicklungsprozesses* und *Verwebung interdisziplinärer Entwicklungsaktivitäten* genutzt wurde. Weiterhin beschreibt [Jos] eine Formalisierung der Risiko- und Gefahrenanalyse der ISO DIS 26262, die für die automatisierte Einflussanalyse in Kapitel 8 eingebunden wurde.

9.3 Leistungen dieser Arbeit

Wie es bereits bei der Betrachtung verwandter Arbeiten angedeutet ist, liefert die vorliegende Arbeit ein Konzept für die *Formalisierung des Entwicklungsprozesses* und *Verwebung interdisziplinärer Entwicklungsaktivitäten*. Dieses bettet sich in das Forschungsfeld *Virtual Engineering* der *Nationalen Roadmap Embedded Systems* [ZVE09] ein, insbesondere in den langfristigen Forschungsbereich *Multi-Domain Engineering*.

Dazu wurde eine neuartige Methode zur *Verwebung interdisziplinärer Entwicklungsaktivitäten* beschrieben, die einem Entwickler zielorientiert externes Wissen zur Verfügung stellt, um ihn bei der Bewältigung von Produkt- und Prozesskomplexität zu unterstützen. Dabei spielt die Formalisierung von Domänenwissen eine entscheidende Rolle. Dieses formalisierte Wissen kann in verschiedenen Entwicklungsprojekten wiederverwendet werden und kann sich somit schnell amortisieren. Insbesondere liegt einem Entwickler eine vollständigere, sowie werkzeuggestützt analysierbare Wissens- und Entscheidungsbasis vor. Die kann er nutzen, um schneller und besser planbar Produkte von höherer Qualität zu entwickeln. Neben der reduzierten Entwicklungszeit können also auch Kosten reduziert werden (weniger Reklamationen).

An der Methodik besonders als neu hervorzuheben ist die Integration eines hohen Formalisierungsgrades mit einer großen Offenheit. Der hohe Formalisierungsgrad macht die angesprochene Werkzeugunterstützung möglich. Die große Offenheit sorgt für eine gute Wiederverwendung, auch in heterogenen Werkzeuglandschaften. Darüber hinaus sorgt sie für die explizite Formulierung von Fakten der Domäne und des Produktes, was die Interaktion zwischen den Disziplinen verbessert.

Die *Formalisierung des Entwicklungsprozesses* ist wesentlicher Bestandteil des in dieser Arbeit vorgeschlagenen Vorgehensmodells. Es wurde aufgezeigt, wie die Formalisierung und Verknüpfung von Domänenwissen genutzt werden kann, den Entwurfsraum eines Entwicklers um für ihn unbekannte und nicht sinnvolle Bereiche einzugrenzen und einen geeigneten Arbeitsplan abzuleiten. Dazu wird aus Domäneneigenschaften eines Produktes der Einfluss auf andere Teile des Produktes und Prozesses inferiert. Dieses zielt auf die Reflexion eigener Entwurfsentscheidungen, umso schneller und besser planbar zu einem geeigneten Entwurf zu gelangen. Dieser erfüllt die Anforderungen von Kunden und der Domäne. Weiterhin konnte demonstriert werden, wie die abstrakte Modellierung auf Standards basierend mit einer serviceorientierten Implementierung zusammenspielen kann.

Technisch wurde das Vorgehensmodell aus Kapitel 5 mit Hilfe von Ontologien modelliert, eine Transformation auf BPEL, NuSMV und HTML wurde in XSLT implementiert. Die Einflussanalyse aus Kapitel 6 wurde in OWL modelliert und liegt als Apache Ant Task vor. Gleiches gilt für die generische Implementierung mit OWL, SWRL und SPARQL. Die Codetransformationen aus den in-Vehicle Service Languages wurden in XSLT implementiert und ist im Rahmen der DOMINION Laufzeitumgebung verwendbar. Somit konnten erste Erfahrungen in der technischen Anwendung gesammelt werden. Besonders wichtig zu erwähnen sind hierbei die technischen Grenzen und konzeptionellen Eigenschaften der Inferenz auf Basis der *Open World Assumption*. Technische Beschränkungen können bei sehr großen Modellen dazu führen, dass eine direkte Inferenz nicht praktikabel erscheint. An diesen Stellen wurde im Rahmen der prototypischen Umsetzung auf SWRL und SPARQL zurückgegriffen. Konzeptionell kann die Offenheit bedeuten, dass mehr Modellierungsaufwand notwendig ist als bei geschlossenen Modellen. Jedoch wird durch das Fehlen impliziter Annahmen bei der Modellierung eine flexible, langfristige und qualitätsgesicherte Nutzung möglich.

Die Formalisierung von Domänenwissen bezüglich funktionaler Sicherheit im Automobilbereich (ISO DIS 26262) auf Grundlage der OWL bildet weiterhin die Grundlage des *Certification Support*

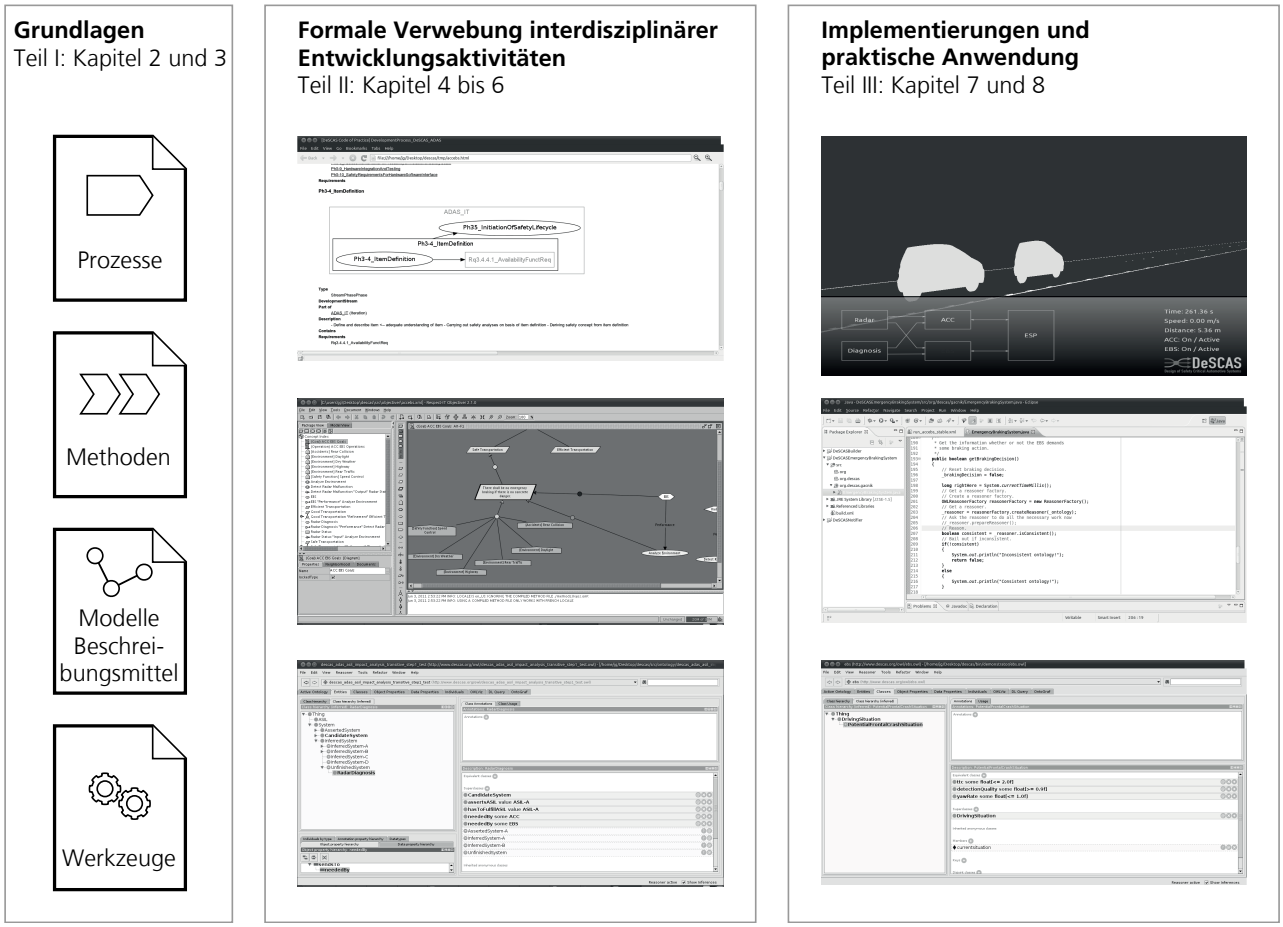

Abbildung 9-1: Leistungen und praktische Umsetzungen dieser Arbeit.

Process im europäischen Projekt CESAR [JHK+10]. Dabei wurde auch eine technische Abbildung der OWL Modelle auf das RMM und SPEM implementiert.

Insgesamt ist besonders zu betonen, dass nicht nur ein Konzept erarbeitet wurde, sondern dieses auch mit einer Vielzahl integrierter technischer Lösungen prototypisch umgesetzt und in verschiedenen Projekten eingesetzt wurde (siehe auch Abbildung 9-1 bzw. Abbildung C-1).

9.4 Anknüpfungspunkte und Ausblick

Im Rahmen dieser Arbeit wurde eine Methode vorgestellt, die formalisiertes Domänenwissen nutzt, um Entwicklern externes Wissen zielorientiert zur Verfügung zu stellen. Die Bewertung des im zur Verfügung stehenden Wissens bezüglich seiner Entwurfsalternativen obliegt jedoch nach wie vor dem Entwickler selbst. Werden etwa quantitative monetäre Größen einbezogen, könnten dem Entwickler optimierte Entwurfsvorschläge präsentiert werden.

Eine sinnvolle Erweiterung der beschriebenen Methode also wäre die Nutzung des formalen Rahmens als Randbedingung in einer *Optimierungsaufgabe*, die basierend auf dem formalisierten Wissen der Entwicklungsstränge den Lösungsraum nach einer optimalen Lösung durchsucht. Dazu ist für den Zeitraum der Optimierung eine *Schließung* des offenen Domänenwissens notwendig. Bei Verwendung der *Open World Assumption* wäre der Lösungsraum *offen*, also unendlich und für eine deterministische Optimierung unentscheidbar. Weiterhin muss die vor allem *qualitative* Einflussanalyse um *quantitative* Faktoren erweitert werden, um Entwurfsalternativen automatisiert bewerten zu können. Solche Qualitätsmodelle sind ebenfalls Teil des formalisierten Domänenwissens und etwa im Kontext der ISO DIS 26262 auch normativ bekannt. Bezüglich der

Optimierung scheinen *lokale* Optimierungsverfahren erfolgversprechend (z.B. Simplex-Verfahren). Ergänzend könnten auch *globale* Verfahren (z.B. evolutionäre Algorithmen) eingesetzt werden.

Wie im Anwendungsbeispiel angedeutet, bietet sich eine Evaluation der Nutzung der OWL als Beschreibungsmittel für eine sicherheitskritische Logik im Kontext von *Evolution und Individualisierung von AAS* an, dabei vor allem für Architekturen im C2X Umfeld. Hier kann die Offenheit im Zusammenhang mit der formalen Spezifikation genutzt werden.

Weitere Optionen ergeben sich in *integrativer Form*. So würde eine Einbindung weiterer Werkzeuge und Methoden sinnvoll sein. Es erscheint ebenfalls lohnenswert, die beschriebene Methode als einen Rahmen für Forschungen im Bereich der virtuellen Zertifizierung zu begreifen. Denn insbesondere im Kontext des Entwurfs sicherheitskritischer AAS liegen durch Normen und Gesetze vergleichsweise klare und harte Kriterien vor, die für eine Formalisierung von Domänenwissen herangezogen werden können. Im Gegensatz dazu wäre jedoch auch eine konkrete Umsetzung im Entwicklungsstrang *Faktor Mensch* interessant, hier liegen vergleichsweise weiche Anforderungen vor. Aufgrund des hohen generischen Anteils der Methodik kann sich auch eine Anwendung in anderen Anwendungsfeldern (z.B. Medizintechnik) lohnen.

Teil V

Anhang

A Glossar

AAS (Assistenz- und Automationssystem) Die AAS umfassen im Rahmen dieser Arbeit sowohl die Menge der *Advanced Driver Assistance Systems* (ADAS) [Eur06b], als auch die Menge der *in-Vehicle Information Systems* (IVIS) [Eur06b]. Weiterhin fallen darunter auch solche Systeme, die über den unmittelbaren Fahrzeugkontext hinaus unterstützen, wie ein *Travel Assistance System* (TAS) [GHHK08], sowie Systeme, die auf *Car-2-X Communication Systems* (C2X) [CAR07] basieren.

ACC (Adaptive Cruise Control) Ein ACC ist ein Abstandsregeltempomat. Somit ist es in der Lage, neben der Geschwindigkeit eines Fahrzeugs einen gewünschten Abstand zu einem eventuell vorausfahrenden Fahrzeug einzuregeln. Zur sensorischen Erfassung des Fahrzeugumfeldes wird in der Regel ein Radar verwendet [Bos02]. Das ACC ist im Rahmen der ISO 15622 [ISO02b] international standardisiert. Das ACC und dessen Erweiterung dient in dieser Arbeit als Anwendungsbeispiel.

AGHL (Automotive Generic Hazard List) Die AGHL [Jos10] wurde aus Unfall- und Systemcharakteristiken mit Blick auf Gebrauch und vorhersehbaren Missbrauch des Systems ermittelt. Dabei sind kausale Zusammenhänge zwischen Schaden und technischer Ursache erfasst. Somit können bestimmte Eingriffe eines AAS bezüglich der von ihnen ausgehenden Gefahr klassifiziert werden. Die AGHL ist in die Werkzeugkette der vorliegenden Arbeit eingebunden.

ASIL (Automotive Safety Integrity Level) Der ASIL ist im Rahmen der ISO DIS 26262 [ISO09] definiert. Er gibt für einen Entwicklungsgegenstand an, welche Anforderungen bzw. Methoden angewendet werden müssen, um ein bestimmtes tolerierbares Risiko zu erreichen. Es gibt vier ASIL Level: A, B, C, D. Dabei ist D der Level mit den meisten und am stärksten fordernden Anforderungen, A ist der am wenigsten fordernde Level.

BPEL (Business Process Execution Language) BPEL ist als XML Derivat standardisiert [JE07]. Dabei handelt es sich um eine Sprache zur Beschreibung von Geschäftsprozessen und Workflows, deren Aktivitäten durch WSDL basierte Dienste implementiert werden. Eine solche Komposition von Diensten wird auch als Orchestrierung bezeichnet. Diese Arbeit nutzt BPEL zur Beschreibung von ausführbaren Instanzen eines Vorgehensmodells.

Controllability Kontrollierbarkeit ist in der ISO DIS 26262 [ISO09] und im RESPONSE 3 CoP [Eur06b] definiert. Sie gibt an, in welchem Umfang ein Fahrer mit seiner Reaktion in der Lage ist, einen durch einen Systemausfall drohenden Schaden abzuwenden. Dieser Parameter spielt eine wichtige Rolle im Rahmen der Risikoanalyse im Kontext der ISO DIS 26262.

CESAR (Cost-Efficient methods and processes for SAfety Relevant embedded systems) Seit März 2009 existiert das europäische Forschungsprojekt *Cost-Efficient methods and processes for SAfety Relevant embedded systems* (CESAR), in dessen Rahmen insgesamt 55 Partner zusammenarbeiten. Dabei werden Methoden und Werkzeuge für industrielle Anwendungen entwickelt. Diese sollen die Entwicklung hochzuverlässiger eingebetteter Systeme ermöglichen, welche den gestiegenen gesellschaftlichen Bedarf an Mobilität decken.

DeSCAS (Design of Safety-Critical Automotive Systems) Das virtuelle Institut DeSCAS[1] verbindet seit März 2007 das Institut für Verkehrssystemtechnik im Deutschen Zentrum für Luft- und Raumfahrt, das Forschungszentrum Sicherheitskritische Systeme der Carl von Ossietzky Universität Oldenburg und das Institut für Verkehrssicherheit und Automatisierungstechnik

[1] DeSCAS: `http://www.descas.org/`

der Technischen Universität Braunschweig. DeSCAS wird von der Helmholtz Gemeinschaft gefördert und hat das Ziel, einen sicherheitsorientierten und gleichzeitig menschzentrierten Entwicklungsprozess zu definieren.

ESB (Enterprise Service Bus) Ein ESB ist eine wesentliche Komponente im Rahmen einer SOA Implementierung. Er setzt die verteilte Interaktion und Kommunikation zwischen verschiedenen Dienstimplementierung um. Ein ESB wird im Rahmen dieser Arbeit zur Umsetzung flexibler und offener Anwendungen genutzt.

Exposure Die Exposure ist im Kontext der ISO DIS 26262 [ISO09] und RESPONSE 3 CoP [Eur06b] definiert. Sie gibt an, dass eine Situation vorliegt, die potentiell gefährlich ist und zu einem Schaden führen kann. Dieser Parameter spielt eine wichtige Rolle im Rahmen der Risikoanalyse im Kontext der ISO DIS 26262.

GOMS (Goals, Operators, Methods and Selection rules) Die GOMS Methoden wurden ursprünglich von [CMN83] erwähnt. Diese kommen aus der Psychologie, insbesondere aus der Interaktion von Menschen und Computern. Sie können als Instrument in der Aufgabenanalyse verwendet werden, welche das Ziel verfolgt, die Aufgaben eines menschlichen Operators zu identifizieren und zu strukturieren. GOMS versucht dabei, die Ziele eines Nutzers zu formulieren, diese in Teilziele zu dekomponieren und zu zeigen, wie diese durch eine Interaktion erreicht werden. Die Verwebung eines GOMS Modells mit einem KAOS Modell wird im Rahmen dieser Arbeit zur Verdeutlichung der Modellverwebung benutzt.

GSN (Goal Structuring Notation) Die GSN wurde mit dem Fokus auf der Unterstützung von *Safety Cases* an der Universität York entwickelt [KW04]. Im Speziellen soll die GSN, eine grafische Notation zur Argumentation, die Lücke zwischen Sicherheitsanforderungen und assoziierten Nachweisen schließen. Dafür werden unter Verwendung einer Zielnotation Sicherheitsziele zu Teilzielen und Lösungen dekomponiert. Diese Dekomposition umfasst die Notation von Strategien, welches die Argumentation unterstützt.

IEC 61508 Die 1998 eingeführte IEC 61508 [IEC03b] thematisiert die funktionale Sicherheit elektrischer, elektronischer und elektronisch programmierbarer Systeme. Sie stellt eine herstellerübergreifende Referenz für Vorgehensmodelle zur Entwicklung sicherheitskritischer Anwendungen zur Verfügung. Solche Referenzprozesse sind insofern besonders wichtig, da sie bei der Zulassung sicherheitskritischer Systeme als Bezug für den Stand der Forschung und Technik dienen.

ISO DIS 26262 Die sich aktuell in der Normierung befindende ISO 26262 stellt einen sogenannten Applikationsstandard (oder Derivat) der IEC 61508 für den Automobilbau dar; sie liegt im Sommer 2010 als *Draft International Standard* [ISO09] vor. Dieser Standard zur funktionalen Sicherheit verfeinert die Konzepte der Basisnorm und passt sie an die Domäne des Automobilbaus an. Diese Arbeit verwendet die ISO DIS 26262 als Quelle für die Modellierung von Domänenwissen zur funktionalen Sicherheit.

KAOS (Knowledge Acquisition in autOmated Specification) Die KAOS Methode entstand 1990 aus einer Kooperation der Universität von Oregon und der Universität von Louvain in Belgien. Letztere entwickelt die Methode bis heute weiter [Res07]. Dabei existieren neben der Methode auch ein Beschreibungsmittel (das KAOS Metamodell) und ein Werkzeug (Objectiver). Nach [Hei05] ist KAOS besonders gut für die Dokumentation bzw. Spezifikation von Anforderungen geeignet. KAOS wird als Requirements Engineering Methode im Rahmen dieser Arbeit exemplarisch angewendet.

Kripke Struktur Eine Kripke Struktur ist ein formales Beschreibungsmittel, welches in vielen Modelchecking Werkzeugen verwendet wird. In diesem Zusammenhang können Kripke Strukturen zur Systembeschreibung genutzt werden. Weiterhin existieren eine Reihe automatischer

Übersetzer, um andere Beschreibungsmittel (z.B. Programmiersprachen) in Kripke Strukturen zu überführen. Im Rahmen dieser Arbeit werden Kripke Strukturen zur formalen Modellierung von Prozessabläufen verwendet.

LTL (Lineare temporale Logik) LTL kann genutzt werden, um Spezifikationen auszudrücken. Diese können im Rahmen des Modelchecking verwendet werden, um eine automatische Verifikation eines Modells gegen dessen Spezifikation durchzuführen. Ein solches Modell kann etwa durch eine Formalisierung in Kripke Strukturen gegeben sein. In dieser Arbeit werden lineare temporale Logikformeln im Rahmen der Prozessverifikation genutzt.

Modelchecking Modelchecking dient der Verifikation einer Systembeschreibung (Modell) gegen eine Spezifikation (Formel). Im Rahmen dieser Arbeit wird ein Modelchecker verwendet, um bestimmte Eigenschaften eines Vorgehensmodells zu prüfen. Dazu werden Kripke Strukturen (Modell) und lineare temporale Logik (Formel) verwendet.

MOF (Meta Object Facility) MOF ist bei der OMG standardisiert [OMG06] und beschreibt eine spezielle Metadaten-Architektur. MOF unterscheidet dabei vier Modellebenen: Metametamodelle M_3, Metamodelle M_2, Modelle M_1 und konkrete Daten M_0. Das MOF Schema wird in dieser Arbeit zur Strukturierung von Vorgehensmodellen genutzt.

ODM (Ontology Definition Metamodel) Das ODM ist bei der OMG standardisiert [OMG09b] und umfasst die Verknüpfung verschiedener Modelle zur Definition von Domänenwissen: OWL, RDF, UML, CL und Topic Maps. Das ODM wird in dieser Arbeit zur Verknüpfung von Metamodellen und Ontologien herangezogen.

OMG (Object Management Group) Die OMG beschäftigt sich insbesondere mit der Definition von Integrationsstandards für Informationstechnologien. Dazu gehören die in dieser Arbeit genutzten Standards MOF und ODM.

Ontologie Eine Ontologie ist eine explizite Spezifikation einer Konzeptualisierung [Gru95] und wird von mehreren Partnern geteilt. Ontologien im Kontext der Informationstechnik haben ihre Wurzeln insbesondere im Feld der Wissensrepräsentation künstlicher Intelligenz. Solch eine Repräsentation muss vor allem automatisiertes Schlussfolgern (Inferenz) ermöglichen. Ontologien in Form von OWL werden in dieser Arbeit zur offenen Formalisierung von Domänenwissen verwendet.

OWA (Open World Assumption) Die OWA sagt aus, dass der Wahrheitsgehalt einer Aussage unabhängig davon ist, ob ein einzelner Beobachter sie als wahr erkennt. Das bedeutet, dass keine Schlussfolgerungen möglich sind, die sich auf *implizite* Annahmen stützen. Die der OWL zugrundeliegende Logik basiert auf der OWA. Das Gegenteil der OWA ist die *Closed World Assumption*. Die OWA ist wesentlicher Bestandteil einer offenen Formalisierung im interdisziplinären Kontext, wie sie im Rahmen dieser Arbeit beschrieben wird.

OWL (Web Ontology Language) Die OWL ist eine Sprache zur Modellierung von Ontologien. Sie ist ein offener Standard [DS04] vom W3C. Inzwischen liegt die OWL in der Version 2 vor [PPS08]. Die OWL 2 erweitert die Semantik [Sch09a] von RDF. Weiterhin hat sie eine direkte Semantik im Sinne einer Beschreibungslogik [MPSG09], einer Prädikatenlogik erster Ordnung [HKS06]. Der Vorteil der direkten Semantik auf Basis einer Beschreibungslogik ist, dass in OWL beschriebene Ontologien einer formalen Betrachtungsweise zugeführt werden können. So kann etwa das Modell auf Konsistenz geprüft werden. Weiterhin ist logisches Schlussfolgern (Deduktion, Inferenz) möglich. Dieses erfolgt unter Verwendung sogenannter (semantischer) *Reasoner*. Die OWL wird in dieser Arbeit zur Formalisierung von Domänenwissen herangezogen.

OWL-S Bei OWL-S [W3C04] vom W3C handelt es sich um eine spezielle Ontologie, die den Bereich von Diensten im Internet fokussiert. Die OWL-S umfasst dabei sehr konkrete Teile, die sich explizit auf eine Dienstespezifikation in der WSDL beziehen. Darüber hinaus umfasst sie

eine abstrahierte Sicht auf den Dienst. Das macht es möglich, in WSDL spezifizierte Dienste mit anderen Domänen und Modellen zu verknüpfen. Die OWL-S wird in dieser Arbeit genutzt, um die Implementierungsebene von Diensten mit der Modellierung von Domänenwissen zu verknüpfen.

RDF (Resource Description Framework) RDF ist beim W3C standardisiert [MM04] und kann insbesondere zur Modellierung von Metadaten genutzt werden. OWL Modelle erweitern RDF. RDF ist eine Kerntechnologie des *Semantic Web*. In den Implementierungen im Rahmen dieser Arbeit werden OWL Ontologien auf der Basis von RDF serialisiert und abgelegt.

Reasoner Ein Reasoner ist eine Software, die es ermöglicht, aus einer Menge Fakten oder Axiomen logische Konsequenzen zu deduzieren (Inferenz). Für entscheidbare Untermengen der OWL kann ein solcher Reasoner verwendet werden. Im Rahmen dieser Arbeit wird ein Reasoner für das Schlussfolgern auf formalisiertem Domänenwissen verwendet.

RESPONSE 3 CoP Im Bezug auf die Normierung der ISO DIS 26262 wurde innerhalb des EU Projektes PreVENT vor allem der Aspekt der *controllability* (Kontrollierbarkeit) und dessen Nachweis im Leitfaden RESPONSE 3 CoP (Code of Practice) näher beleuchtet [Eur06b, Eur06a]. Weist man eine hohe Kontrollierbarkeit des Systems nach, müssen andere technische Maßnahmen zur Risikominderung nicht durchgeführt werden. Der RESPONSE 3 CoP gibt ebenfalls ein Prozessmodell vor, welches sich an einem Wasserfallmodell als Vorgehensmodell orientiert. Für diese Arbeit wird RESPONSE 3 CoP als Quelle für Domänenwissen zum Faktor Mensch einbezogen.

Semantic Web Das *Semantic Web* umfasst eine Menge Methoden, Beschreibungsmittel und Werkzeuge, die die maschinelle Verarbeitung semantischer Informationen (Bedeutung) im Rahmen des Internet ermöglichen sollen. Zu dieser Menge gehören beispielsweise die OWL, RDF, SWRL und SPARQL. Der Begriff des *Semantic Web* wird durch das W3C geprägt. Diese Arbeit nutzt in hohem Maße Standards, die im Kontext des *Semantic Web* definiert sind.

Severity Die Severity ist im Kontext der ISO DIS 26262 [ISO09] und RESPONSE 3 CoP [Eur06b] definiert. Sie gibt an, wie gefährlich eine Situation ist, also welcher Schaden entstehen kann. Dieser Parameter spielt eine wichtige Rolle im Rahmen der Risikoanalyse im Kontext der ISO DIS 26262.

SOA (Service-Oriented Architecture) SOA ist ein Software-Architekturmuster. Es zeichnet sich durch eine starke (Geschäfts-)Prozessorientierung und lose Kopplung sowie einfache Wiederverwendbarkeit einzelner Dienste aus. Einfachere Dienste können zu abstrakteren Diensten *orchestriert* werden. Für diese Arbeit wird SOA als Architekturmuster angewendet.

SOAP (Simple Object Access Protocol) SOAP ist ein beim W3C standardisiertes Netzwerkprotokoll [BEK+00], welches die Übertragung von Daten sowie den entfernten Funktionsaufruf unterstützt. SOAP ist ein XML Derivat und nutzt auf der Anwendungsschicht vor allem HTTP. SOAP wird im Kontext serviceorientierter Architekturen zur Umsetzung der Kommunikation und Interaktion zwischen Diensten genutzt. Für den Einsatz flexibler AAS wird im Rahmen dieser Arbeit SOAP als Netzwerkprotokoll genutzt.

SPARQL (SPARQL Protocol and RDF Query Language) SPAQRL ist eine graphenbasierte Abfragesprache für RDF. SPARQL ist beim W3C standardisiert [PS08]. Die Verwendung von SPARQL Abfragen kann als Ergänzung oder Alternative zu logikbasierter Inferenz im Rahmen der Nutzung von OWL herangezogen werden. SPAQRL ist eine wichtige Technologie im Kontext des *Semantic Web*. Für einzelne Schritte der im Rahmen dieser Arbeit entstandenen Werkzeugkette wird SPARQL zur Modelltransformation genutzt.

SWRL (Semantic Web Rule Language) SWRL wird bei W3C standardisiert [HPSB+04]. SWRL kombiniert eine entscheidbare Untermenge der OWL mit regelbasierten Sprachkonzepten. So

können alternativ zu Axiomen in OWL Inferenzregeln in SWRL formuliert werden. SWRL ist eine wichtige Technologie im Kontext des *Semantic Web*. Im Rahmen dieser Arbeit ist eine Werkzeugkette entstanden, die die Nutzung von SWRL ermöglicht.

VPEL (In-Vehicle Process Execution Language) VPEL ist im Rahmen dieser Arbeit als Teilmenge von BPEL definiert. Die Einschränkungen ergeben sich aus Überlegungen zur Echtzeitfähigkeit und deterministischem Verhalten.

VSD (In-Vehicle Schema Definitions) Im Rahmen dieser Arbeit wird VSD als Teilmenge von XSD definiert. Die Modifikationen ergeben sich aus Betrachtungen zur Echtzeitfähigkeit und deterministischem Verhalten.

VSDL (In-Vehicle Service Description Language) VSDL ist im Rahmen dieser Arbeit aus der WSDL abgeleitet.

WSDL (Web Services Description Language) WSDL ist beim W3C standardisiert [CCMW01]. Es ist eine generische Beschreibungssprache für Dienste im Kontext von SOA. WSDL nutzt XSD zur Beschreibung von Datenstrukturen, die der Dienst über seine Schnittstellen einbindet. In WSDL beschriebene Dienst können durch BPEL zu abstrakteren Diensten orchestriert werden. WSDL wird in dieser Arbeit zur Beschreibung von Softwareimplementierungen verwendet.

W3C (World Wide Web Consortium) Das W3C ist ein Gremium, welches eine Standardisierung von Internettechnologien vornimmt. Dazu gehören die in dieser Arbeit genutzten Technologien XML, XSD, XSLT, WSDL, RDF, OWL-S, OWL, SWRL, SPARQL und SOAP.

XSD (eXtensible Schema Definition) XSD ist beim W3C standardisiert [FW04, TBMM04, BM04]. Es bietet ein Schema zur Definition von XML Derivaten. Es dient somit als Metamodell für konkrete XML Datenstrukturen. XSD wird beispielsweise durch WSDL und BPEL zur Definition von Datenstrukturen genutzt. Im Rahmen dieser Arbeit kommt XSD zur Modellierung von Datenstrukturen zum Einsatz.

XML (eXtensible Markup Language) XML ist eine Sprache zur Darstellung hierarchisch strukturierter Textdaten und ist beim W3C standardisiert [BPSM$^+$06]. XML ist ein wichtiges Basisformat, um Daten zwischen verschiedenen Softwarewerkzeugen auszutauschen. XML wird in dieser Arbeit als Basisformat zum Import und Export von Modellen genutzt.

XSLT (eXtensible Stylesheet Language Transformations) XSLT ist beim W3C standardisiert [Cla99]. Es dient der Definition von Transformationen von XML Dokumenten. Ziel einer solchen Transformation kann ein anderer XML Dialekt oder ein Textdokument sein. Im Rahmen dieser Arbeit wird XSLT zur Transformation von XML Dokumenten zu anderen Dokumenten genutzt. Weiterhin wird es zur Codegenerierung (Text) eingesetzt.

B Notation

B.1 Kripke Strukturen

Kripke Strukturen sind ein formales Beschreibungsmittel, welches in vielen Modelchecking Werkzeugen verwendet wird. Modelchecking dient der Verifikation einer Systembeschreibung (Modell) gegen eine Spezifikation (Formel). Kripke Strukturen können zur Systembeschreibung genutzt werden, temporallogische Formeln (siehe Abschnitt B.2) für die Spezifikation. Weiterhin existieren eine Reihe automatischer Übersetzer, um andere Beschreibungsmittel (z.B. Programmiersprachen) in Kripke Strukturen zu überführen.

Kripke-Strukturen mit totaler Transitionsrelation Die Kripke-Struktur $K = (S, R, L, I)$ umfasst neben einer Menge von Zuständen (states)

$$S \tag{B.1}$$

Relationen (relations), die hier der totalen Transitionsrelation (jeder Zustand hat mindestens einen Nachfolger, gegebenenfalls reflexiv)

$$R \subseteq S \times S \tag{B.2}$$

unterliegen. Dazu kommen atomare Aussagen (atomic propositions) AP, die zu der Menge aller Aussagen (propositions) P zusammengefasst werden. Diese werden durch die sogenannten Beschriftungsfunktionen (labels)

$$L : S \to P(AP) \tag{B.3}$$

verwendet, um den Zuständen eine Menge an Aussagen zuzuordnen, die im jeweiligen Zustand gelten. Dabei existiert eine Menge an Zuständen

$$I \subseteq S \tag{B.4}$$

welche die Anfangszustände (initial states) markieren.

B.2 Lineare temporale Logik (LTL)

Lineare temporale Logik kann genutzt werden, um Spezifikationen auszudrücken. Diese können im Rahmen des Modelchecking verwendet werden, um eine automatische Verifikation eines Modells gegen dessen Spezifikation durchzuführen. Ein solches Modell kann etwa durch eine Formalisierung in Kripke Strukturen (siehe Abschnitt B.1) gegeben sein. Eine LTL Formel kann über einer unendlichen Sequenz von Aussagen (ϕ, ψ) oder einer einzigen Position i auf dem (temporalen) Pfad ausgewertet werden. Dabei existieren eine Reihe von einstelligen und zweistelligen Operatoren, siehe auch Tabelle B-1.

Zweistellige Operatoren sind *Until* (U) und *Release* (R).

Für $\phi U \psi$ muss die Aussage ϕ von der aktuellen Position an solange gelten, bis die Aussage ψ wahr wird. Dabei gilt ψ an der aktuellen oder einen nachfolgenden Position.

Für $\phi R \psi$ muss ϕ gelten, bis ψ wahr wird. Wird ψ nicht wahr, muss ϕ für immer gelten.

Einstellige Operatoren sind *Next* (X), *Finally* (F) und *Globally* (G). Für $X\phi$ muss ϕ an der nächsten Position gelten. Für $F\phi$ muss ϕ irgendwann auf dem folgenden Pfad gelten, für $G\phi$ muss ϕ auf dem kompletten folgenden Pfad gelten.

Name	Syntax
Zweistellige Operatoren	
Until	$\phi U \psi$
Release	$\phi R \psi$
Einstellige Operatoren	
Next	$X\phi$
Finally	$F\phi$
Globally	$G\phi$

Tabelle B-1: Modale Operatoren der linearen temporalen Logik.

Name	Syntax	Semantik	Symbol
Elemente			
Nichtnegative Ganzzahl	n	$n \in \{0, 1, \ldots\}$	$\mathcal{AL}$
Konzept	C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$\mathcal{AL}$
Rolle	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	$\mathcal{AL}$
Einfache Rolle	S	$S^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	$\mathcal{R}$
Instanz	c	$c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$	$\mathcal{AL}$

Tabelle B-2: Elemente einer Beschreibungslogik (zusammengestellt nach [HKS06, BMNPS03]).

B.3 Beschreibungslogik

Die meisten Beschreibungslogiken sind entscheidbare Untermengen der Prädikatenlogik erster Stufe. Die in diesem Abschnitt dargestellten Tabellen umfassen eine Definition der im Rahmen dieser Arbeit verwendeten Beschreibungslogik. Die Tabellen sind dabei nach einem identischen Schema aufgebaut. Der *Name* bezeichnet und beschreibt das jeweilige Sprachelement. *Syntax* und *Semantik* kennzeichnen die Repräsentation und mathematische Bedeutung der Sprachelemente. Das *Symbol* gibt an, zu welcher Kategorie ein Element gehört. Eine Kombination dieser Kategorien ergibt dabei eine bestimmte Form und Ausdrucksstärke einer Beschreibungslogik. So wird in dieser Arbeit die OWL 2 verwendet, welche sich auf die sogenannte Beschreibungslogik $\mathcal{SROIQ}$ [HKS06] stützt. Die OWL 1 hingegen basierte auf $\mathcal{SHOIN}(\mathcal{D})$ [HPS03].

Die basalen *Elemente* einer Beschreibungslogik sind in Tabelle B-2 dargestellt. Positive ganze Zahlen n werden im Rahmen der Beschreibungslogik vor allem für die Beschreibung von Kardinalitäten genutzt. Das Konzept (Klasse) C, die Rolle (Property) R und die Instanz c eines Konzepts sind die zentralen Elemente einer Beschreibungslogik. Formal werden diese durch eine Interpretation $\mathcal{I}$ beschrieben. Diese besteht aus einer nicht-leeren Menge $\Delta^{\mathcal{I}}$ und einer Interpretationsfunktion, die jedem Konzept eine Menge $\Delta^{\mathcal{I}}$ und jeder Rolle eine zweistellige Relation $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ zuweist. Dabei setzt sich die Menge $\Delta^{\mathcal{I}}$ aus den Instanzen c der modellierten Domäne zusammen. Eine Domäne kann weiter konkretisiert werden, etwa zur Modellierung konkreter Datentypen Δ_D. In einem solchen Fall werden neben Datentypen auch die Beziehungen zwischen Instanzen (der Konzepte) und Instanzen von Datentypen definiert. Die formale Modellierung einer konkreten Domäne ähnelt der einer abstrakten Domäne. Zur einfacheren Darstellung werden die Aspekte der konkrete Domäne Δ_D hier ausgelassen, können bei [HPS03] nachgelesen werden. Die einfache Rolle S ist Teil der OWL 2. Die Einführung der einfachen Rolle ist der Entscheidbarkeit von $\mathcal{SROIQ}$ geschuldet. Eine einfache Rolle unterliegt einigen Einschränkungen bezüglich ihres Ausdrucks gegenüber einer vollen Rolle [HKS06]. Für die Klassifikation (Inferenz) ist eine Formalisierung bestimmter Merkmale von Konzepten, Rollen

Name	Syntax	Semantik	Symbol
Konzeptkonstruktoren			
Universelles Konzept	$\top$	$\Delta^{\mathcal{I}}$	$\mathcal{AL}$
Null Konzept	$\bot$	$\emptyset$	$\mathcal{AL}$
Konjunktion	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$	$\mathcal{AL}$
Wertrestriktion	$\forall R.C$	$\{c_1 \mid \forall c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}} \rightarrow c_2 \in C^{\mathcal{I}}\}$	$\mathcal{AL}$
Eing. Existenz	$\exists R.\top$	$\{c_1 \mid \exists c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}}\}$	$\mathcal{AL}$
Disjunktion	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$	$\mathcal{U}$
Komplement	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$	$\mathcal{C}$
Volle Existenz	$\exists R.C$	$\{c_1 \mid \exists c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}} \wedge c_2 \in C^{\mathcal{I}}\}$	$\mathcal{E}$
Kardinalität I	$\geq n\ R$	$\{c_1 \mid \#\{c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}}\} \geq n\}$	$\mathcal{N}$
Kardinalität II	$\leq n\ R$	$\{c_1 \mid \#\{c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}}\} \leq n\}$	$\mathcal{N}$
Kardinalität III	$= n\ R$	$\{c_1 \mid \#\{c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}}\} = n\}$	$\mathcal{N}$
Q. Kardinalität I	$\geq n\ S.C$	$\{c_1 \mid \#\{c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}} \wedge c_2 \in C^{\mathcal{I}}\} \geq n\}$	$\mathcal{N}$
Q. Kardinalität II	$\leq n\ S.C$	$\{c_1 \mid \#\{c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}} \wedge c_2 \in C^{\mathcal{I}}\} \leq n\}$	$\mathcal{N}$
Q. Kardinalität III	$= n\ S.C$	$\{c_1 \mid \#\{c_2.\langle c_1, c_2\rangle \in R^{\mathcal{I}} \wedge c_2 \in C^{\mathcal{I}}\} = n\}$	$\mathcal{N}$
Nominal	N	$N^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ mit $\lvert N^{\mathcal{I}}\rvert = 1$	$\mathcal{O}$
Reflexives Konzept	$\exists S.\mathrm{Self}$	$\{c_1 \mid \langle c_1, c_1\rangle \in S^{\mathcal{I}}\}$	$[\mathcal{R}]$

Tabelle B-3: Konstruktoren von Konzepten in einer Beschreibungslogik (zusammengestellt nach [HKS06, BMNPS03]).

und Instanzen notwendig. Hierzu existieren die *Konzeptkonstruktoren* zur Beschreibung von Konzepten und die *Rollenkonstruktoren* zur Beschreibung von Rollen. Schließlich werden die Konstruktoren in *Axiomen* verwendet, die dann zur Klassifikation (Inferenz) genutzt werden.

Wichtige Elemente sind die *Konzeptkonstruktoren* (siehe Tabelle B-3). Die ermöglichen es, die Eigenschaften eines bestimmten Konzeptes formal zu fassen. Dadurch, dass Konzepte spezifische formale Eigenschaften erhalten, wird eine Konsistenzprüfung und Klassifikation erst möglich. Im Falle der OWL, welche die *Open World Assumption* zugrunde legt, wirken solche Konzeptkonstruktoren *beschränkend* auf das modellierte Konzept (Klasse). Im einfachsten Falle handelt es sich um das universelle Konzept $\top$ (Tautologie) oder um das leere Konzept $\bot$ (Widerspruch). Da Konzepte mit Mengen assoziiert sind, gelten für sie die Mengenoperatoren Konjunktion $C_1 \sqcap C_2$, Disjunktion $C_1 \sqcup C_2$ und Negation beziehungsweise Komplement $\neg C$. Über Rollen können weitere Restriktionen formuliert werden, etwa die eingeschränkte ($\exists R.\top$) oder volle Existenz ($\exists R.C$). Im ersten Fall muss eine Rolle R erfüllt werden, im zweiten Fall muss diese außerdem durch ein spezifisches Konzept C gefüllt werden. Die Existenzaussagen können durch Wertrestriktionen ($\forall R.C$) weiter verfeinert werden. Dabei wird gefordert, dass die Rolle R nur von einem spezifischen Konzept C eingenommen werden darf. Die gemeinsame Anwendung von Existenz- und Wertrestriktionen kann im Falle der OWL für die Formulierung von *Schließungsaxiomen* verwendet werden. Über diese kann durch die *Open World Assumption* bedingte Offenheit der Logik bei Bedarf eingeschränkt (geschlossen) werden. Die Verwendung von Kardinalität ermöglicht es zudem, zulässige Werte für die Häufigkeit bestimmter Rollen von Konzepten zu fordern.

Analog zu Konzeptkonstruktoren existieren eine Reihe von *Rollenkonstruktoren*, die in Tabelle B-4 zusammengefasst sind. Im allgemeinsten Fall handelt es sich um die universelle Rolle U, die der Tautologie bei den Konzepten entspricht. Ebenfalls vorhanden sind die Mengenoperatoren über zweistelligen Relationen, und zwar die Konjunktion $R_1 \sqcap R_2$, Disjunktion $R_1 \sqcup R_2$ sowie die Negation (Komplement) $\neg R$. Für Rollen können darüber hinaus inverse Rollen R^- existieren.

Name	Syntax	Semantik	Symbol
Rollenkonstruktoren			
Universelle Rolle	U	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	$\mathcal{AL}$
Konjunktion	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$	$\mathcal{AL}$
Disjunktion	$R_1 \sqcup R_2$	$R_1^{\mathcal{I}} \cup R_2^{\mathcal{I}}$	$\mathcal{AL}$
Komplement	$\neg R$	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \backslash R^{\mathcal{I}}$	$\mathcal{C}$
Inverse Rollen	R^-	$\{\langle r_2, r_1\rangle \mid \langle r_1, r_2\rangle \in R^{\mathcal{I}}\}$	$\mathcal{I}$

Tabelle B-4: Konstruktoren von Rollen in einer Beschreibungslogik (zusammengestellt nach [HKS06, BMNPS03]).

Name	Syntax	Semantik	Symbol
Axiome: Konzepte (Tbox)			
Konzeptäquivalenz	$C_1 \equiv C_2$	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$	$\mathcal{AL}$
Konzepthierarchie	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$	$\mathcal{AL}$
Axiome: Rollen (Rbox)			
Rollenäquivalenz	$R_1 \equiv R_2$	$R_1^{\mathcal{I}} = R_2^{\mathcal{I}}$	$\mathcal{AL}$
Rollenhierarchie	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$	$\mathcal{H}$
Transitivität	$\mathrm{Tra}(R)$	$\langle c_1, c_2\rangle \in R^{\mathcal{I}} \wedge \langle c_2, c_3\rangle \in R^{\mathcal{I}} \rightarrow \langle c_1, c_3\rangle \in R^{\mathcal{I}}$	$\mathcal{S}$
Symmetrie	$\mathrm{Sym}(R)$	$\langle c_1, c_2\rangle \in R^{\mathcal{I}} \rightarrow \langle c_2, c_1\rangle \in R^{\mathcal{I}}$	$\mathcal{R}$
Reflexivität	$\mathrm{Ref}(R)$	$\{\langle c_1, c_1\rangle \mid c_1 \in \Delta^{\mathcal{I}}\} \subseteq R^{\mathcal{I}}$	$\mathcal{R}$
Irreflexivität	$\mathrm{Irr}(S)$	$S^{\mathcal{I}} \cap \{\langle c_1, c_1\rangle \mid c_1 \in \Delta^{\mathcal{I}}\} = \emptyset$	$\mathcal{R}$
Disjunktheit	$\mathrm{Dis}(S_1, S_2)$	$S_1^{\mathcal{I}} \cap S_2^{\mathcal{I}} = \emptyset$	$\mathcal{R}$
Fakten (Abox)			
Konzeptzuweisung	$C(c_1)$	$c_1^{\mathcal{I}} \in C^{\mathcal{I}}$	$\mathcal{AL}$
Rollenzuweisung	$R(c_1, c_2)$	$\langle c_1^{\mathcal{I}}, c_2^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$	$\mathcal{AL}$
Neg. Rollenzuweisung	$\neg R(c_1, c_2)$	$\langle c_1^{\mathcal{I}}, c_2^{\mathcal{I}}\rangle \ni R^{\mathcal{I}}$	$\mathcal{AL}$
Gleichheit	$c_1 = c_2$	$c_1^{\mathcal{I}} = c_2^{\mathcal{I}}$	$\mathcal{F}$
Ungleichheit	$c_1 \neq c_2$	$c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$	$\mathcal{F}$

Tabelle B-5: Axiome und Fakten innerhalb einer Beschreibungslogik (zusammengestellt nach [HKS06, BMNPS03]).

Über die Konstruktoren für Rollen und Konzepte können *Axiome* und *Fakten* aufgestellt werden, diese sind in Tabelle B-5 zusammengestellt. Diese greifen Konzepte und Rollen auf, die durch Konzeptkonstruktoren beziehungsweise Rollenkonstruktoren beschrieben sind. Sie formulieren notwendige und hinreichende Bedingungen für die Klassifikation (Inferenz). Bezüglich der Terminologie existieren jeweils Hierarchie (notwendige Bedingung) und Äquivalenz (notwendige und hinreichende Bedingung). Diese sind auch für Rollen vorhanden. Für Rollen existieren weitere Eigenschaften, die gefordert werden können. Dazu gehören die Transitivität $\mathrm{Tra}(R)$, Symmetrie $\mathrm{Sym}(R)$, Reflexivität $\mathrm{Ref}(R)$, Irreflexivität $\mathrm{Irr}(S)$ sowie die Disjunktheit $\mathrm{Dis}(S_1, S_2)$. Darüber hinaus kann durch Zusicherungen die Zugehörigkeit von Instanzen zu Konzepten oder deren Rollen ausgedrückt werden. Auch Gleichheit beziehungsweise Ungleichheit können als Tatsache aufgenommen werden.

C Abbildungen zur Werkzeugkette

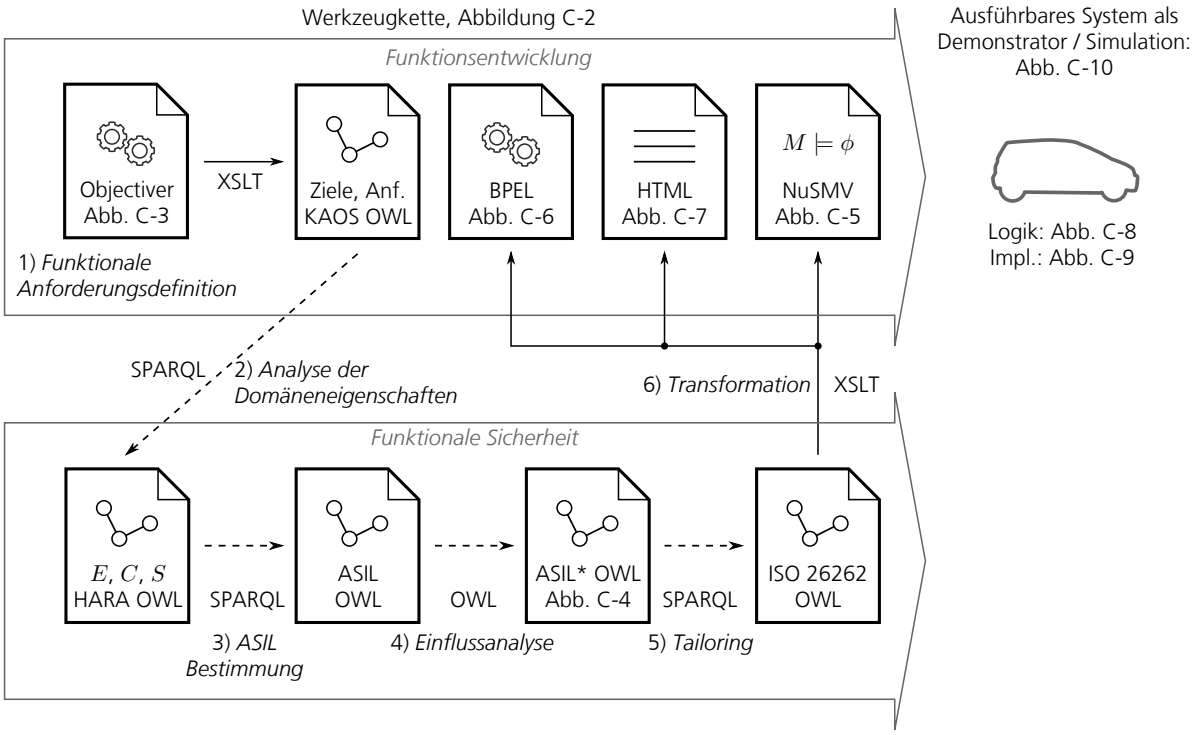

Abbildung C-1: Übersicht der Abbildungen zur Werkzeugkette und dem ausführbaren Demonstrator (analog zu Abbildung 8-2).

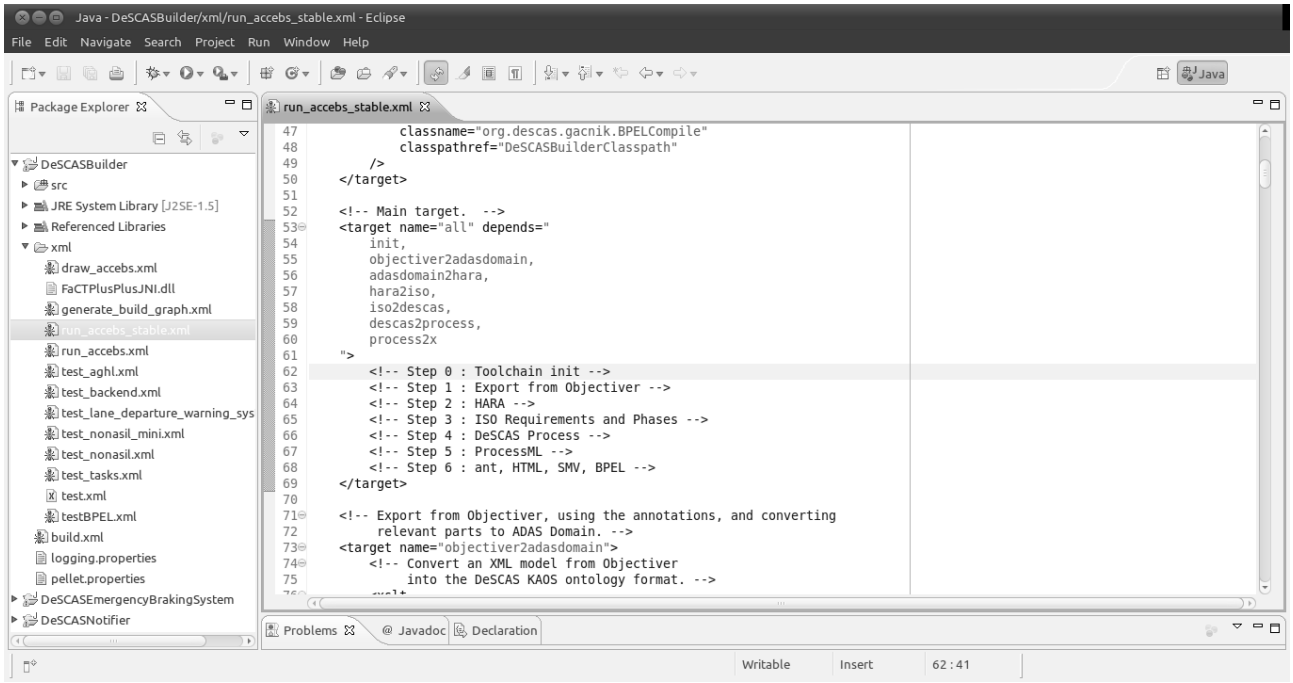

Abbildung C-2: Anwendungsbeispiel ACC / EBS: Werkzeugkette in *Eclipse*, basierend auf *Apache Ant*.

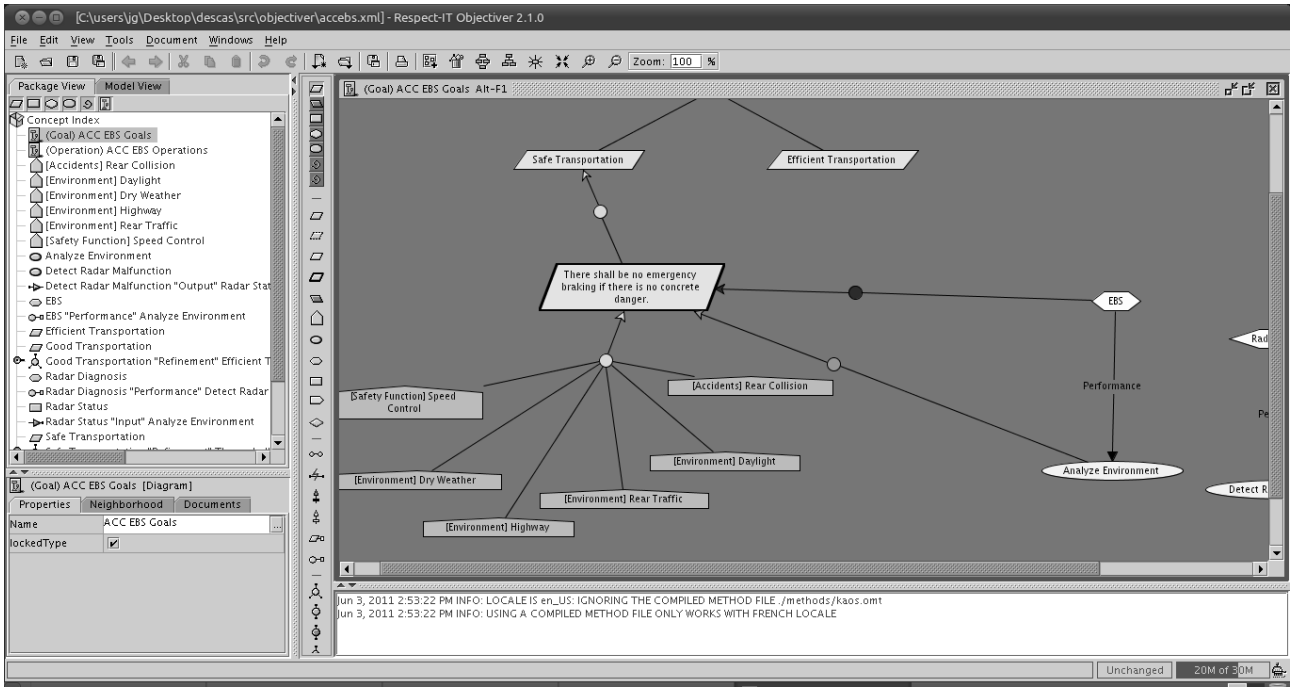

Abbildung C-3: Anwendungsbeispiel ACC / EBS: Umsetzung der funktionalen Anforderungsspezifikation in *Objectiver*, welches die KAOS Methode implementiert.

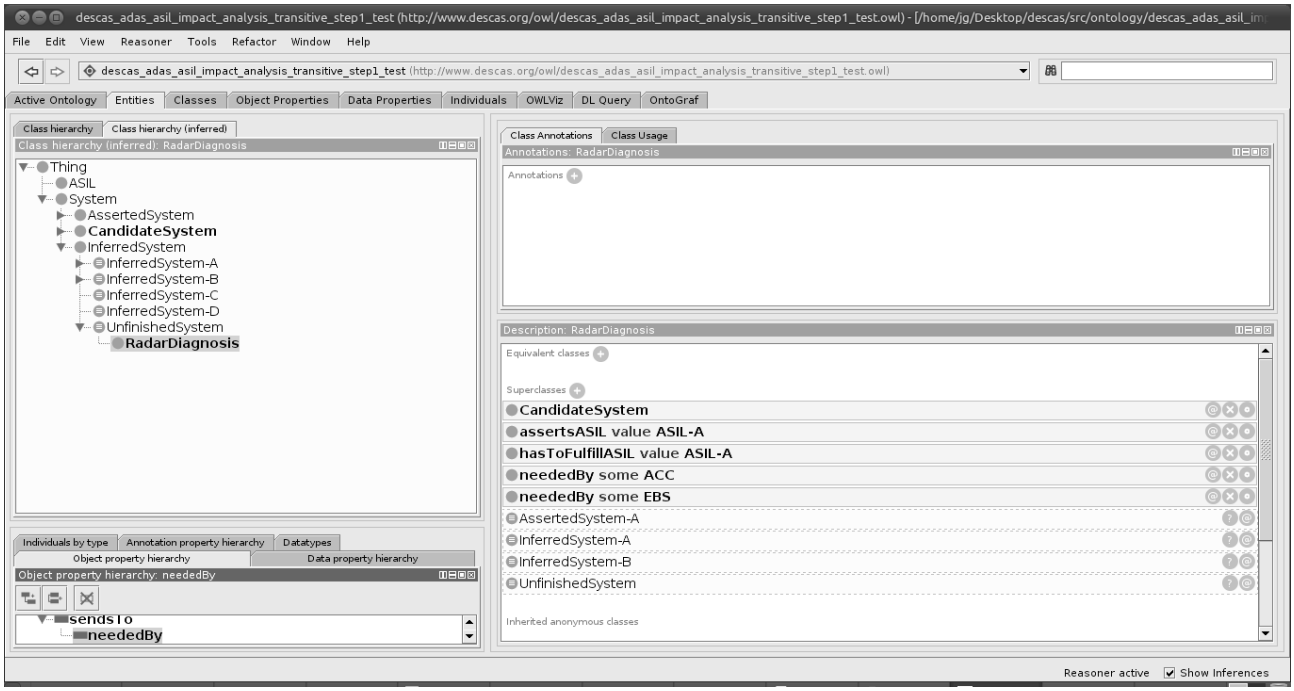

Abbildung C-4: Anwendungsbeispiel ACC / EBS: Umsetzung der Einflussanalyse in *Protégé*, einem Werkzeug zur Modellierung von OWL Ontologien.

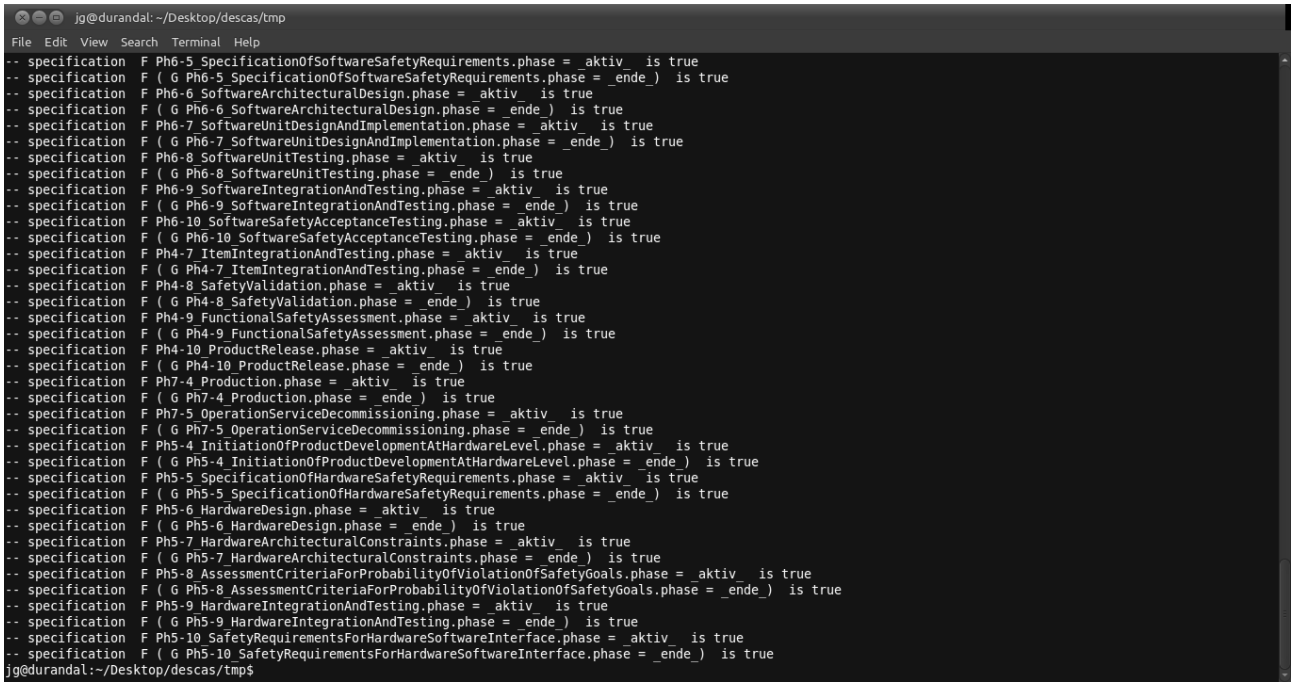

Abbildung C-5: Anwendungsbeispiel ACC / EBS: Verifikation des Prozessmodells mit dem Modelchecker *NuSMV*.

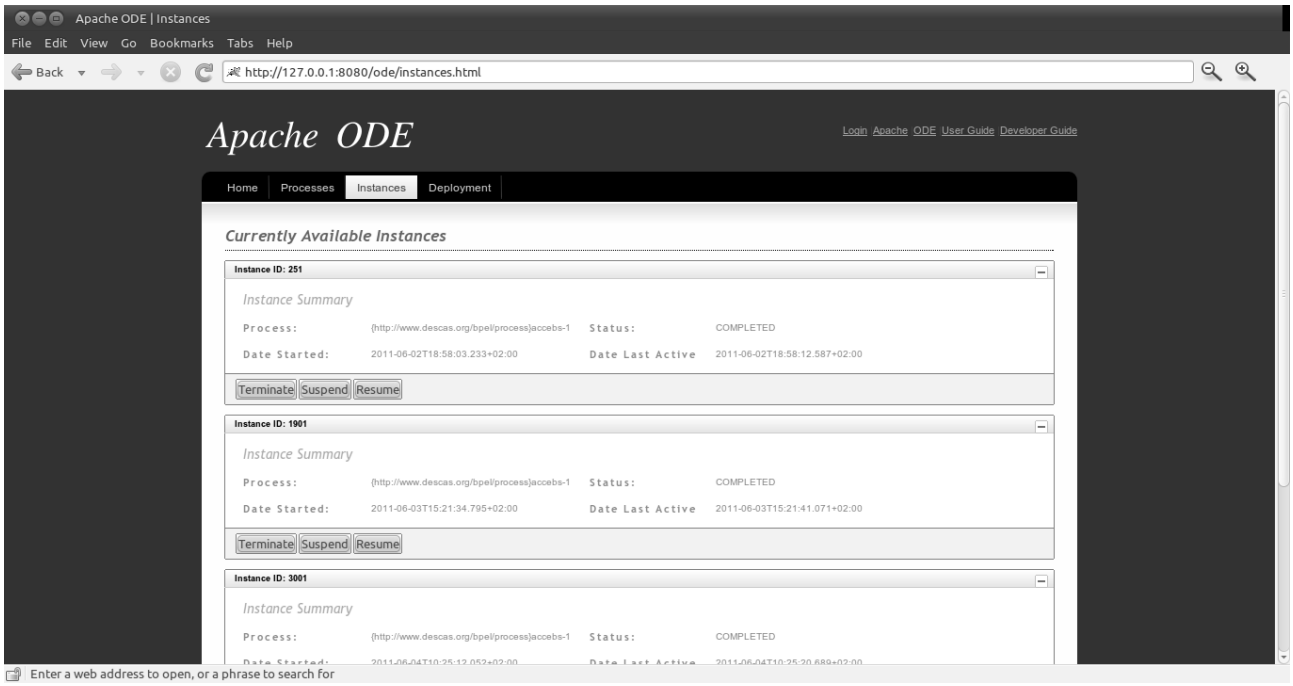

Abbildung C-6: Anwendungsbeispiel ACC / EBS: Instanziierung des maßgeschneiderten Prozesses im BPEL Prozessmanager *Apache Ode*.

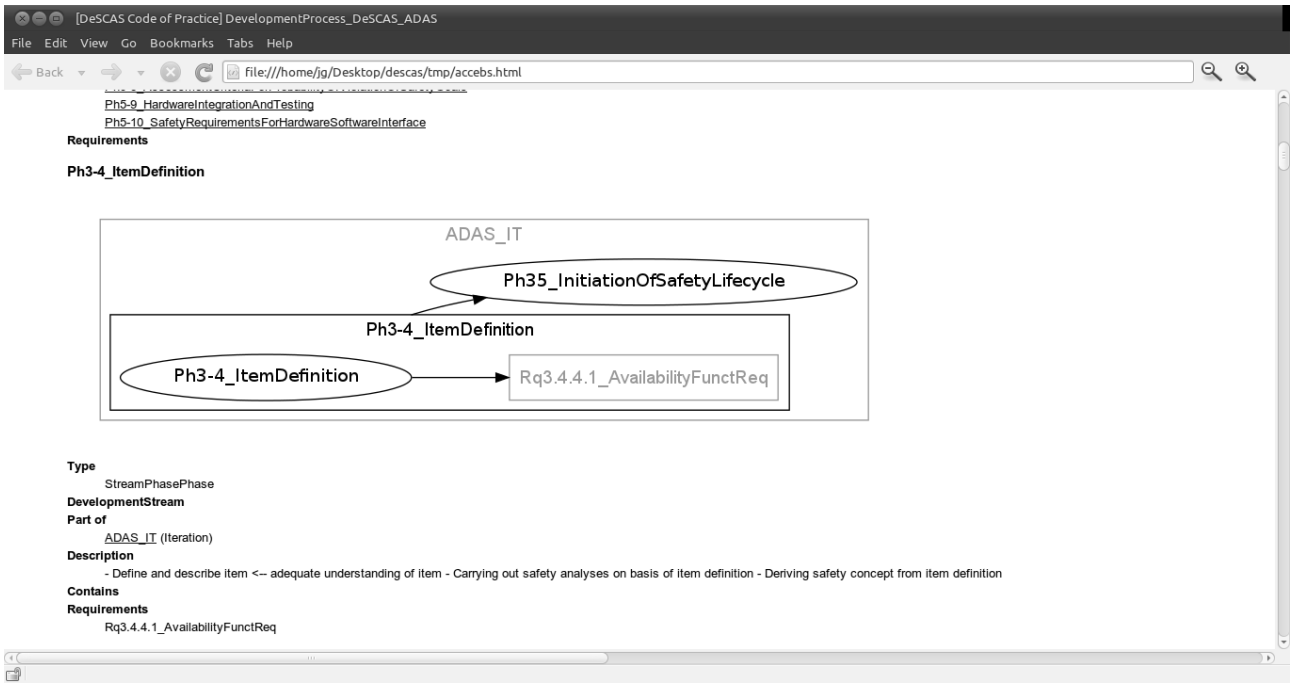

Abbildung C-7: Anwendungsbeispiel ACC / EBS: Generierung von HTML Dokumentation des maßgeschneiderten Prozesses.

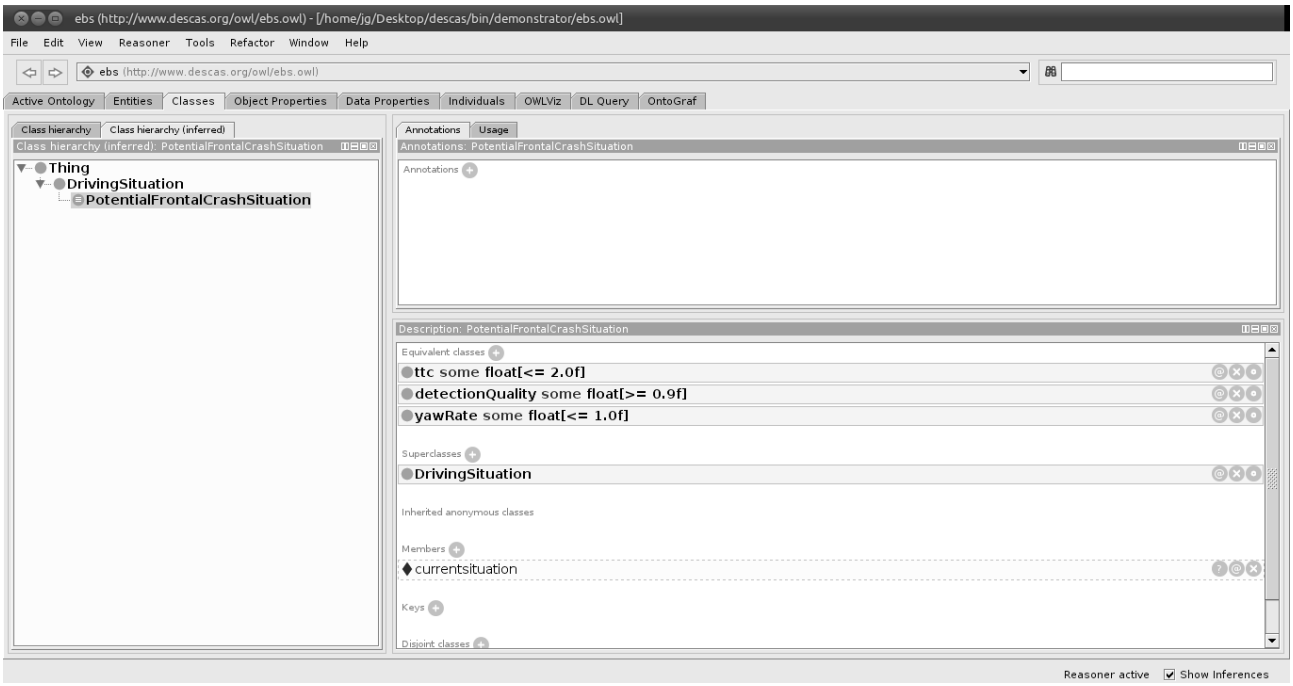

Abbildung C-8: Anwendungsbeispiel ACC / EBS: Modellierung der EBS Entscheidung bezüglich Bremsmanöver als Ontologie in *Protégé*.

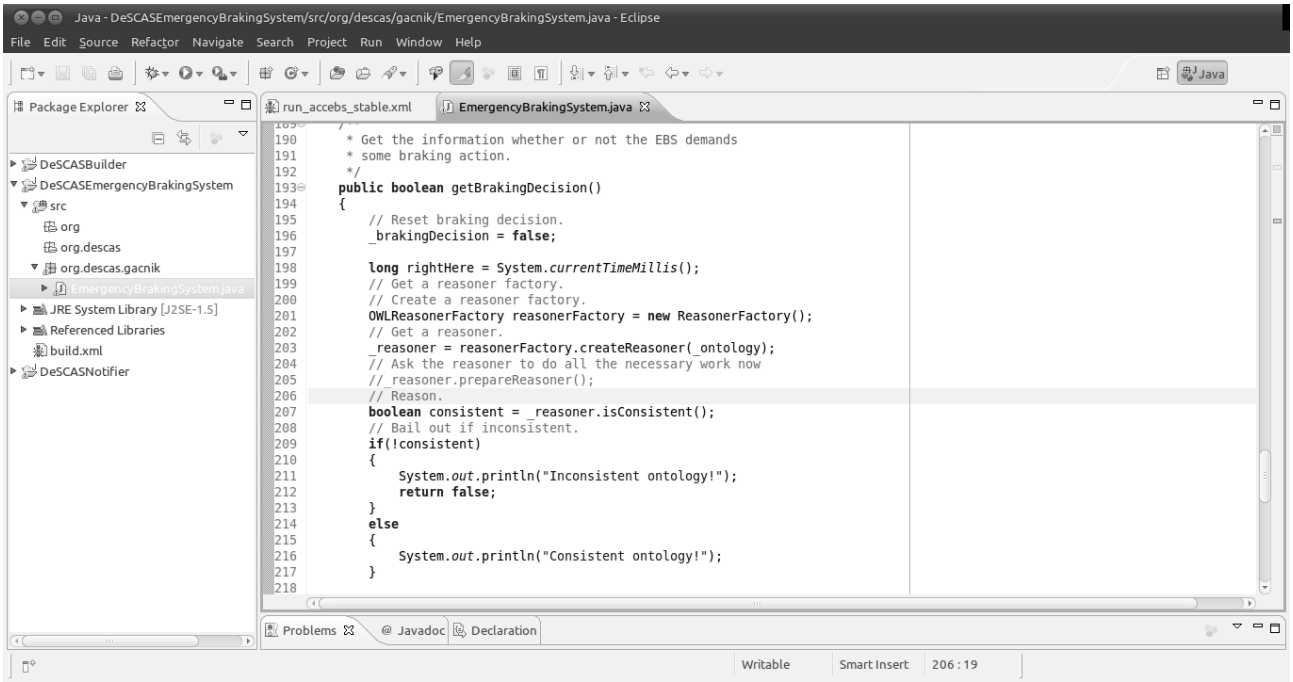

Abbildung C-9: Anwendungsbeispiel ACC / EBS: Implementierung des EBS mit Hilfe des Java Reasoners *HermiT* unter *Eclipse*.

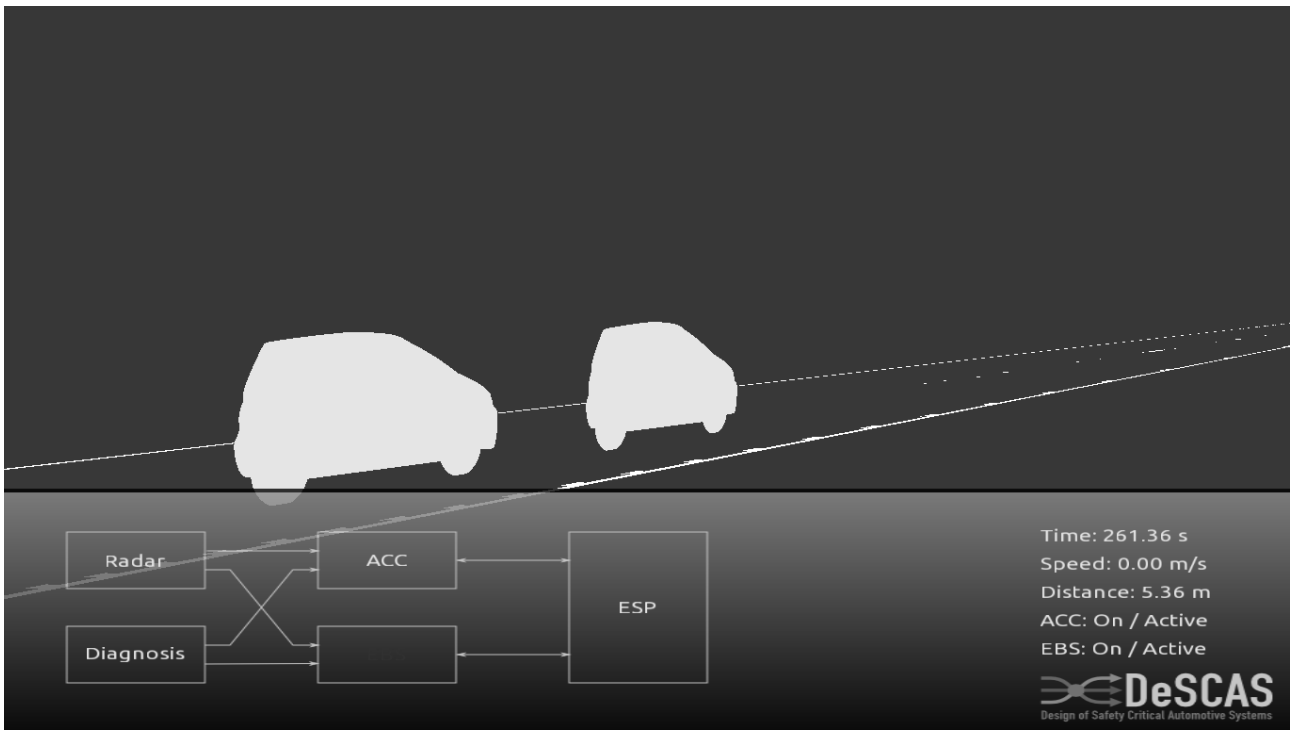

Abbildung C-10: Anwendungsbeispiel ACC / EBS: Softwaresimulation der über die in-Vehicle Service Languages realisierten Implementierung in DOMINION.

Abkürzungs- und Symbolverzeichnis

AAS	Assistenz- und Automationssystem
Abox	Assertional Box
ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistance System
AGHL	Automotive Generic Hazard List
ASIL	Automotive Safety Integrity Level
AUTOSAR	AUTomotive Open System ARchitecture
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
C	Controllability
C2X	Car-2-X Communication
CESAR	Cost-Efficient methods and processes for SAfety Relevant embedded systems
CL	Common Logics
CMN	Card-Morgan-Newell
CoP	Code of Practice
CPM	Cognitive-Perceptual-Motor
CTL	Computation Tree Logic
CWA	Closed World Assumption
DARE	Domain Analysis and Reuse Environment
DeSCAS	Design of Safety-Critical Automotive Systems
DIS	Draft International Standard
DSDM	Dynamic Systems Development Method
DSL	Domain-Specific Language
E	probability of Exposure
EBS	Emergency Braking System

ECU	Electronic Control Unit
ER	Entity Relationship
ESB	Enterprise Service Bus
ESP	Electronic Stability Program
FAST	Family-oriented Abstraction, Specification, and Translation
FDD	Feature Driven Development
FMECA	Failure Mode, Effects, and Criticality Analysis
FORM	Feature-Oriented Reuse Method
FTA	Fault Tree Analysis
GOMS	Goals, Operators, Methods and Selection rules
GSN	Goal Structuring Notation
HARA	Hazard Analysis and Risk Assessment
HTA	Hierarchical Task Analysis
IVIS	in-Vehicle Information System
KAOS	Knowledge Acquisition in autOmated Specification
KLM	Keystroke-Level-Model
KobrA	Komponentenbasierte Anwendungsentwicklung
LTL	Linear Temporal Logic
M2T	Model to Text
MDD	Model-Driven Development
MILO	MId-Level Ontology
MMIO	Memory Mapped I/O
MOF	Meta Object Facility
NGOMSL	Natural GOMS Language
OASIS	Organization for the Advancement of Structured Information Standards
OCL	Object Constraint Language
ODM	Ontology Definition Metamodel
OMG	Object Management Group

OWA	Open World Assumption
OWL	Web Ontology Language
PLUS	Product Line UML-based Software engineering
QL	Query Language
QVT	Query View Transformation
RAD	Rapid Application Development
Rbox	Role Box
RDF	Resource Description Framework
RL	Rule Language
RMM	Requirements Meta-Model
RUP	Rational Unified Process
S	Severity of potential harm
SCR	Software Cost Reduction
SGT	Sub-Goal Template Method
SIL	Safety Integrity Level
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SP	Subproject
SPARQL	SPARQL Protocol and RDF Query Language
SPEM	Software & Systems Process Engineering Meta-Model
SUMO	Suggested Upper Merged Ontology
SWRL	Semantic Web Rule Language
SysML	Systems Modeling Language
TAS	Travel Assistance System
Tbox	Terminological Box
TDD	Test Driven Development
TTA	Tabular Task Analysis
TTC	Time To Collision

UDP	User Datagram Protocol
UML	Unified Modeling Language
VPA	Verbal Protocol Analysis
VPEL	In-Vehicle Process Execution Language
VSD	In-Vehicle Schema Definitions
VSDL	In-Vehicle Service Description Language
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XP	Extreme Programming
XSD	eXtensible Schema Definition
XSLT	eXtensible Stylesheet Language Transformations
XT	eXtreme Tailoring

Literaturverzeichnis

[ABB+01] Atkinson, Colin, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jurgen Wust und Jorg Zettel: *Component-Based Product Line Engineering with UML*. Addison-Wesley Professional, November 2001.

[AF98] Artale, Alessandro und Enrico Franconi: *A Temporal Description Logic for Reasoning about Actions and Plans*. Journal of Artificial Intelligence Research, 9:463 bis 506, 1998.

[AFC+01] Artale, Alessandro, Enrico Franconi, Milenko Mosurovi C, Frank Wolter und Michael Zakharyaschev: *Temporal Description Logics*. In: *Handbook of Time and Temporal Reasoning in Artificial Intelligence*, Seite 96 bis 105. MIT Press, 2001.

[Aga07] Agarwal, Sudhir: *Formal Description of Web Services for Expressive Matchmaking*. Dissertation, Universität Karlsruhe, 2007.

[Ant97] Antón, Ana I.: *Goal Identification and Refinement in the Specification of Software-Based Information Systems*. Dissertation, Georgia Institute of Technology, June 1997.

[AZW06] Aßmann, Uwe, Steffen Zschaler und Gerd Wagner: *Ontologies, Metamodels, and the Model-Driven Paradigm*. In: *Ontologies for Software Engineering and Technology*, 2006.

[BBL05] Baader, Franz, Sebastian Brandt und Carsten Lutz: *Pushing the EL Envelope*. In: *Proceedings of the 19th Joint International Conference on Artificial Intelligence (IJCAI 2005)*, 2005.

[Bec99] Beck, Kent: *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, Oktober 1999.

[Bec02] Beck, Kent: *Test-Driven Development By Example*. Addison-Wesley Longman, November 2002.

[BEF+10] Buschermöhle, Ralf, Heike Eekhoff, Heiko Frommold, Bernhard Josko und Micha Schiller: *SUCCESS - Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland*. Technischer Bericht, OFFIS, 2010.

[Beh04] Behrendt, Heiko: *Multi-, Inter- und Transdisziplinarität - Und die Geografie?* In: Frank Brand, Franz Schaller und Harald Völker (Herausgeber): *Transdisziplinarität. Bestandsaufnahme und Perspektiven*, Seite 115 bis 128, 2004.

[BEK+00] Box, Don, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte und Dave Winer: *Simple Object Access Protocol (SOAP) 1.1*. Technischer Bericht, W3C, 2000. `http://www.w3.org/TR/soap/`.

[Ben04] Beneke, Frank: *Produktentwicklung. Arbeiten in und mit verschiedenen Disziplinen - wozu?* In: Frank Brand, Franz Schaller und Harald Völker (Herausgeber): *Transdisziplinarität. Bestandsaufnahme und Perspektiven*, Seite 79 bis 91, 2004.

[Bjø08] Bjørner, Dines: *From domain to requirements*. In: *Concurrency, Graphs and Models*, Band 5065 der Reihe *Lecture Notes in Computer Science*, Seiten 278--300. Springer, 2008.

[Bjø10] Bjørner, Dines: *Domain engineering*. In: *Formal Methods: State of the Art and New Directions*, Seite 1 bis 42. Springer, 2010.

[BJR99] Booch, Grady, Ivar Jacobsen und James Rumbaugh: *The Unified Software Development Process*. Addision-Wesley Professional, Februar 1999.

[BKPS07] Broy, Manfred, Ingolf H. Krüger, Alexander Pretschner und Christian Salzmann: *Engineering Automotive Software*. Proceedings of the IEEE, 95(2), 2007.

[BM04] Biron, Paul V. und Ashok Malhotra: *XML Schema Part 2: Datatypes*. W3C, 2004. `http://www.w3.org/TR/xmlschema-2/`.

[BMNPS03] Baader, Franz, Deborah L. McGuinness, Daniele Nardi und Peter F. Patel-Schneider (Herausgeber): *The Description Logic Handbook: Theory, implementation, and applications*. Cambridge University Press, 2003.

[BMV07] BMVBS (Bundesministerium für Verkehr, Bau und Stadtentwicklung): *Straßenverkehrsordnung (StVO)*, 2007.

[Boe86] Boehm, Barry: *A Spiral Model of Software Development and Enhancement*. ACM SIGSOFT Software Engineering Notes, 11(4):14 bis 24, August 1986.

[Bos02] *Adaptive Fahrgeschwindigkeitsregelung ACC*. BOSCH Gelbe Reihe. Robert Bosch GmbH, 1st Auflage, April 2002.

[BPSM+06] Bray, T., J. Paoli, C.M. Sperberg-McQueen, E. Maler und F. Yergeau: *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. Technischer Bericht, W3C, http://www.w3.org/TR/REC-xml/, 2006.

[BRS10] Beisel, Daniel, Cornelia Reuß und Eckehard Schnieder: *Automotive Generic Hazard List*. In: *Automatisierungs-, Assistenzsysteme und eingebettete Systeme für Transportmittel (AAET)*, 2010.

[Bun89] Bundesministerium der Justiz: *Gesetz über die Haftung für fehlerhafte Produkte (Produkthaftungsgesetz ProdHaftG)*, Dezember 1989.

[Bun06] Bundesministerium für Bildung und Forschung (BMBF): *Die Hightech-Strategie für Deutschland*, 2006.

[BvK08] Bunse, Christian und Antje von Knethen: *Vorgehensmodelle kompakt*. Spektrum Akademischer Verlag, 2. Auflage, 2008.

[CAR07] CAR 2 CAR Communication Consortium: *Manifesto*, August 2007.

[CC05] Cristani, Matteo und Roberta Cuel: *A survey on ontology creation methodologies*. International Journal on Semantic Web & Information Systems, 1(2):49 bis 69, April bis Juni 2005.

[CCM08] Chung, Paul W.H., Larry Y.C. Cheung und Colin H.C. Machin: *Compliance Flow - Managing the compliance of dynamic and complex processes*. Knowledge-Based Systems, 21(4):332 bis 354, 2008, ISSN 0950-7051.

[CCMW01] Christensen, Erik, Francisco Curbera, Greg Meredith und Sanjiva Weerawarana: *Web Services Description Language (WSDL) 1.1*. W3C, 2001. `http://www.w3.org/TR/wsdl`.

[Che76] Chen, Peter Pin Shan: *The Entity-Relationship Model - Toward a Unified View of Data*. ACM Transactions on Database Systems, Seite 9 bis 36, 1976. `http://csc.lsu.edu/news/erd.pdf`.

[Cla99] Clark, James: *XSL Transformations (XSLT) Version 1.0*. W3C, http://www.w3.org/TR/xslt, 1999.

[CLL99] Coad, Peter, Eric Lefebvre und Jeff De Luca: *Java Modeling In Color With UML: Enterprise Components and Process*, Kapitel 6. Prentice Hall PTR, Juni 1999.

[CMN83] Card, Stuart, Thomas P. Moran und Allen Newell: *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[Deu91] Deutsches Institut für Normung e.V. (DIN): *DIN 19227: Graphische Symbole und Kennbuchstaben für die Prozessleittechnik*, 1991.

[Deu95] Deutsches Institut für Normung e.V. (DIN): *DIN 19226: Regelungstechnik und Steuerungstechnik*, 1995.

[Die97] Die Beauftragte der Bundesregierung für Informationstechnik: *V-Modell 97*, Februar 1997.

[Die09] Die Beauftragte der Bundesregierung für Informationstechnik: *V-Modell XT*, Februar 2009. `http://www.v-modell-xt.de/`, Version 1.3.

[dMBSA08] Miguel, M. A. de, J. F. Briones, J. P. Silva und A. Alonso: *Integration of safety analysis in model-driven software development*. Software, IET, 2(3):260 bis 280, 2008.

[DS04] Dean, Mike und Guus Schreiber: *OWL Web Ontology Language Reference*. Technischer Bericht, W3C, http://www.w3.org/TR/owl-ref/, 2004.

[Eas91] Easterbrook, Steve: *Elicitation of Requirements from Multiple Perspectives*. Dissertation, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, June 1991.

[EFA+99] Emmerich, Wolfgang, Anthony Finkelstein, Stefano Antonelli, Stephen Armitage und Richard Stevens: *Managing standards compliance*. IEEE Trans. Softw. Eng., 25(6):836 bis 851, 1999, ISSN 0098-5589.

[EK08] Emam, Khaled El und A. Günes Koru: *A Replicated Survey of IT Software Project Failures*. IEEE Software, 25(5):84 bis 90, 2008.

[Eur97a] European Committee for Electrotechnical Standardisation (CENELEC): *EN 50126 - Railway Applications: The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*, 1997.

[Eur97b] European Committee for Electrotechnical Standardisation (CENELEC): *EN 50128 - Railway Applications: Software for Railway Control and Protection Systems*, 1997.

[Eur06a] Europe's Information Society: *RESPONSE 3 - Annexes to the Code of Practice for the Design and Evaluation of ADAS*, Oktober 2006.

[Eur06b] Europe's Information Society: *RESPONSE 3 - Code of Practice for the Design and Evaluation of ADAS*, Oktober 2006.

[Eur09] European Telecommunications Standards Institute (ETSI): *ETSI TR 102 638: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions*, 2009. Technical Report, V1.1.1.

[FBH+06] Fennel, Helmut, Stefan Bunzel, Harald Heinecke, Jürgen Bielefeld, Simon Fürst, Klaus Peter Schnelle, Walter Grote, Nico Maldener, Thomas Weber, Florian Wohlgemuth, Jens ruh, Lennart Lundh, Tomas Sandén, Peter Heitkämper, Robert Rimkus, Jean Lefleur, Alain Gilberg, Ulrich Virnich, Stefan Voget, Kenji Nishikawa, Kazuhiro Kaijo, Klaus Lange, Thomas Scharnhorst und Bernd Kunkel: *Achievements and exploitation of the AUTOSAR development partnership*. In: *SAE Convergence*, 2006.

[FF89] Finkelstein, Anthony und Hugo Fuks: *Multi-party specification*, 1989.

[FK05] Frakes, William B. und Kyo Kang: *Software reuse research: Status and future*. IEEE Transactions on Software Engineering, 31:529 bis 536, 2005.

[FPDF98] Frakes, William, Ruben Prieto-Diaz und Christopher Fox: *Dare: Domain analysis and reuse environment*. Annals of Software Engineering, 5:125 bis 141, 1998, ISSN 1022-7091.

[FW04] Fallside, David C. und Priscilla Walmsley: *XML Schema Part 0: Primer*. W3C, 2004. `http://www.w3.org/TR/xmlschema-0/`.

[Gac10] Gacnik, Jan: *Providing Guidance In An Interdisciplinary Model-Based Design Process*. In: *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2010.

[GHH08] Gačnik, Jan, Oliver Häger und Marco Hannibal: *A Service-Oriented System Architecture for the Human Centered Design of Intelligent Transportation Systems*. In: *European Conference on Human Centred Design for Intelligent Transport Systems*, HUMANIST publications, Seite 175 bis 181. HUMANIST, 2008.

[GHHK08] Gačnik, Jan, Marco Hannibal, Oliver Häger und Frank Köster: *Service-Oriented Architecture for Future Driver Assistance Systems*. In: *FISITA World Automotive Congress*, 2008.

[GJA92] Gray, Wayne D., Bonnie E. John und Michael E. Atwood: *The Precis of Project Ernestine or an overview of a validation of GOMS*. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1992.

[GJB+09] Gačnik, Jan, Henning Jost, Daniel Beisel, Jürgen Rataj und Frank Köster: *DeSCAS Design Process Model for Automotive Systems - Development Streams and Ontologies*. In: *Safety-Critical Systems 2009*, Nummer SP-2222 in *Special Publications*. SAE International, 2009.

[GJBK08] Gačnik, Jan, Henning Jost, Daniel Beisel und Frank Köster: *DeSCAS - Design Process for the Development of Safety-Critical Advanced Driver Assistance Systems*. In: Tarnai, Géza und Eckehard Schnieder (Herausgeber): *Formal Methods for Automation and Safety in Railway and Automotive Systems*, 2008.

[GJK+09] Gačnik, Jan, Henning Jost, Frank Köster, Jürgen Rataj, Karsten Lemmer, Werner Damm, Martin Fränzle und Eckehard Schnieder: *DeSCAS - Formale Ontologien zur Verwebung von interdisziplinären Entwicklungsprozessen*. In: *AUTOMATION 2009*, Nummer 2067 in *VDI-Berichte*. VDI Wissensforum GmbH, 2009.

[Gom04] Gomaa, Hassan: *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison-Wesley Professional, Juli 2004.

[Gru95] Gruber, Thomas R.: *Towards principles for the design of ontologies used for knowledge sharing*. International Journal of Human and Computer Studies, 43(5 bis 6):907 bis 928, 1995, ISSN 1071-5819.

[Hei05] Heine, Christian: *Zielorientierte Requirements Engineering Methoden zur Entwicklung von Agentensystemen*. In: *it - Information Technology*, Band 47 (2005) 1. Oldenbourg Verlag, 2005.

[Häg08] Häger, Oliver: *Integrationsarchitektur für den individuellen Verkehrsteilnehmer im globalen Kommunikationskontext*. Diplomarbeit, Fachhochschule Braunschweig/Wolfenbüttel - Fachbereich Informatik, 2008.

[HKS06] Horrocks, Ian, Oliver Kutz und Uli Sattler: *The Even More Irresistible SROIQ*. In: *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*. AAAI Press, 2006.

[Hän08] Hänsel, Frank: *Zur Formalisierung technischer Normen*. Dissertation, Technische Universität Braunschweig, Düsseldorf, 2008, ISBN 9783183787104.

[HPS03] Horrocks, Ian und Peter F. Patel-Schneider: *Reducing OWL Entailment to Description Logic Satisfiability*. Journal of Web Semantics, Seite 17 bis 29, 2003.

[HPSB+04] Horrocks, Ian, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof und Mike Dean: *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, 2004. `http://www.w3.org/Submission/SWRL/`, W3C Member Submission.

[IEC03a] IEC (International Electrotechnical Commission): *IEC 61131: Programmable Controllers Package*, 2003.

[IEC03b] IEC (International Electrotechnical Commission): *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2003.

[IEC05] IEC (International Electrotechnical Commission): *IEC 61499: Function Blocks*, 2005.

[IEE85] IEEE (The Institute of Electrical and Electronics Engineers, Inc.): *IEEE Standard for Binary Floating-Point Arithmetic*, 1985.

[Int08] International Technology Roadmap for Semiconductors: *International Technology Roadmap for Semiconductors 2008 Update*, 2008.

[ISO77] ISO (International Organization for Standardization): *ISO 3833: Road vehicles - Types - Terms and definitions*, 1977.

[ISO01a] ISO (International Organization for Standardization): *ISO/CD 22179: Intelligent Transport Systems - Curve Speed Warning Systems - Performance requirements and test procedures*, 2001. Committee Draft.

[ISO01b] ISO (International Organization for Standardization): *ISO/IEC 9126: Software engineering - Product quality*, 2001.

[ISO02a] *ISO 15623: Transport information and control systems - Forward vehicle collision warning systems - Performance requirements and test procedures*, 2002.

[ISO02b] ISO (International Organization for Standardization): *ISO 15622: Transport information and control systems - Adaptive Cruise Control systems - Performance requirements and test procedures*, 2002.

[ISO03a] ISO (International Organization for Standardization): *ISO 17287: Road vehicles - Ergonomic aspects of transport information and control systems - Procedure for assessing suitability for use while driving*, 2003.

[ISO03b] ISO (International Organization for Standardization): *ISO/IEC 13250: Information technology - SGML applications - Topic maps*, 2003.

[ISO08] ISO (International Organization for Standardization): *Systems and software engineering - System life cycle processes*, 2008.

[ISO09] ISO (International Organization for Standardization): *ISO/DIS 26262: Road Vehicles - Functional Safety*, 2009. Draft International Standard - Baseline 15.

[JE07] Jordan, Diane und John Evdemon: *Web Services Business Process Execution Language Version 2.0*, 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, OASIS Standard.

[JHK+10] Jost, Henning, Stefan Häusler, Silke Köhler, Jan Gačnik, Axel Hahn, Frank Köster und Karsten Lemmer: *Supporting Qualification - Safety Standard Compliant Process Planning and Monitoring*. In: *IEEE International Symposium on Product Compliance Engineering*, 2010.

[JK96] John, Bonnie E. und David E. Kieras: *The goms family of user interface analysis techniques: Comparison and contrast*. ACM Transactions on Computer-Human Interaction, 3(4):320 bis 351, 1996, ISSN 1073-0516.

[Jos] Jost, Henning: *Reasoning on Domain Knowledge and Technical Standards to Support the Development of Safety-Critical Automotive Systems*. Dissertation, Universität Oldenburg. In Bearbeitung.

[Jos10] Jost, Henning: *Automating the Risk and Hazard Analysis via General Domain Concepts in Formal Ontologies*. In: *European Safety and Reliability Association (ESREL) - Annual Conference*, 2010.

[Kai08] Kaindl, Herrmann: *What is an Aspect in Aspect-oriented Requirements Engineering?* In: *International Conference on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD)*, 2008.

[KGH08] Köster, Frank, Jan Gačnik und Marco Hannibal: *Serviceorientierung als Zugang zur Strukturierung von in-Vehicle Softwaresystemen*. In: *IMA 2008 - 4. Braunschweiger Symposium Informationssysteme für mobile Anwendungen*, 2008.

[Kie88] Kieras, David: *Handbook of Human-Computer Interaction*, Kapitel Towards a practical GOMS model methodology for user interface design. Elsevier Science Publishers, Amsterdam, The Netherlands, 1988.

[KLD02] Kang, Kyo C., Jaejoon Lee und Patrick Donohoe: *Feature-oriented product line engineering*. IEEE Software, 19(4):58 bis 65, Juli / August 2002.

[KS96] Kotonya, Gerald und Ian Sommerville: *Requirements Engineering with Viewpoints*. Software Engineering Journal, 11:5 bis 18, 1996.

[KW04] Kelly, Tim und Rob Weaver: *The Goal Structuring Notation - A Safety Argument Notation*. In: *Proceedings of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[LK06] Lawrence, Kelvin und Chris Kaler: *OASIS Web Services Security (WSS) TC*. Technischer Bericht, OASIS, 2006. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`.

[LlCPG+11] Löper, Christian, Álvaro Catalá-Prat, Jan Gačnik, Jan Schomerus und Frank Köster: *Vehicle automation in cooperation with v2i and nomadic devices communication*.

In: *2011 14th International IEEE Conference on Intelligent Transportation Systems*, Seiten 650--655, 2011.

[Mar91] Martin, James: *Rapid Application Development*. Macmillan USA, Mai 1991.

[McC04] McCall, Gavin: *MISRA-C:2004 - Guidelines for the use of the C language in critical systems*. Technischer Bericht, MISRA, 2004.

[MCR07] Mascardi, Viviana, Valentina Cordì und Paolo Rosso: *A comparison of upper ontologies*. In: *WOA*, Seite 55 bis 64, 2007.

[MDL87] Mills, H.D., M. Dyer und R.C. Linger: *Cleanroom Software Engineering*. IEEE Software, 4(5):19 bis 25, 1987.

[MED07a] MEDEA Office Association: *CATRENE (Cluster For Application and Technology Research in Europe on NanoElectronics) White Book, Part A: Rationale and Organisation*, 2007.

[MED07b] MEDEA Office Association: *CATRENE (Cluster For Application and Technology Research in Europe on NanoElectronics) White Book, Part B: Applications and Technologies*, 2007.

[MGH+09] Motik, Boris, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue und Carsten Lutz: *OWL 2 Web Ontology Language Profiles*. World Wide Web Consortium (W3C), ktoberOktober 2009. `http://www.w3.org/TR/owl2-profiles/`, W3C Recommendation.

[MHGK03] Mutz, Martin, Michaela Huhn, Ursula Goltz und Carsten Krömke: *Model-Based System Development in Automotive*. In: *World Congress of the Society of Automotive Engineers, SAE Technical Paper Series 2003-01-1017*, 2003.

[MJET09] Maga, Cameila, Nasser Jazdi, Thomas Ehben und Thilo Tetzner: *Domain Engineering - Mehr Systematik im industriellen Lösungsgeschäft*. In: *AUTOMATION 2009*, Nummer 2067 in *VDI-Berichte*, Seite 69 bis 72. VDI Wissensforum GmbH, 2009.

[ML10] Mitschke, Andreas und Neil Loughran: *RE Language Definitions to Formalize Multi-Criteria Requirements V1*, April 2010. CESAR SP2 Deliverable.

[MM99] Manola, Frank und Eric Miller: *XML Path Language (XPath) Version 1.0*. World Wide Web Consortium (W3C), ovemberNovember 1999. `http://www.w3.org/TR/xpath/`, W3C Recommendation.

[MM04] Manola, Frank und Eric Miller: *RDF Primer*. World Wide Web Consortium (W3C), ebruarFebruar 2004. `http://www.w3.org/TR/rdf-primer/`, W3C Recommendation.

[MPSG09] Motik, Boris, Peter F. Patel-Schneider und Bernardo Cuenca Grau: *OWL 2 Web Ontology Language Direct Semantics*. World Wide Web Consortium (W3C), ktoberOktober 2009. `http://www.w3.org/TR/owl2-direct-semantics/`, W3C Recommendation.

[NFG+11] Nothdurft, Tobias, Tobias Frankiewicz, Jan Gačnik, Peter Hecker und Frank Köster: *Reliable information aggregation and exchange for autonomous vehicles*. In: *2011 IEEE 74th Vehicular Technology Conference: VTC2011-Fall*. IEEE, 2011.

[NP01] Niles, Ian und Adam Pease: *Towards a Standard Upper Ontology*. In: *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, Seite 2 bis 9, New York, NY, USA, 2001. ACM, ISBN 1-58113-377-4.

[NSM+08] Noyer, Ulf, Jan Schomerus, Henning Mosebach, Jan Gacnik, Christian Löper und Karsten Lemmer: *Generating high precision maps for advanced guidance support*. In: *IEEE Intelligent Vehicles Symposium 2008*, Eindhoven, Niederlande, Juni 2008. IEEE Intelligent Transportation Systems Society.

[OMG06] OMG (Object Management Group): *Meta Object Facility (MOF) Core Specification*, Januar 2006. `http://www.omg.org/spec/MOF/2.0`, Spezifikation.

[OMG07] OMG (Object Management Group): *OMG Systems Modeling Language (OMG SysML), V1.0*, September 2007.

[OMG08a] OMG (Object Management Group): *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, April 2008. `http://www.omg.org/spec/QVT/1.0`.

[OMG08b] OMG (Object Management Group): *MOF Model to Text Transformation Language, v1.0*, Januar 2008. `http://www.omg.org/spec/MOFM2T/1.0`, Spezifikation.

[OMG08c] OMG (Object Management Group): *Software & Systems Process Engineering Meta-Model Specification*, April 2008. `http://www.omg.org/spec/SPEM/2.0`, Specification.

[OMG09a] OMG (Object Management Group): *Business Process Model And Notation (BPMN)*, Januar 2009. `http://www.omg.org/spec/BPMN/1.2`, Spezifikation.

[OMG09b] OMG (Object Management Group): *Ontology Definition Metamodel*, May 2009. `http://www.omg.org/spec/ODM/1.0`.

[OMG09c] OMG (Object Management Group): *Unified Modeling Language (UML), Superstructure*, Februar 2009. `http://www.omg.org/spec/UML/2.2/Superstructure/PDF/`, Spezifikation.

[OMG10] OMG (Object Management Group): *Object Constraint Language*, Februar 2010. `http://www.omg.org/spec/OCL/2.2`, Spezifikation.

[Pai10a] Pai, Ganesh: *Requirements Engineering State of the Art Survey*, April 2010. CESAR SP2 Deliverable.

[Pai10b] Pai, Ganesh: *Requirements Engineering State of the Practice Survey*, April 2010. CESAR SP2 Deliverable.

[Pet62] Petri, Carl Adam: *Kommunikation mit Automaten*. Dissertation, Universität Bonn, 1962.

[Poh08] Pohl, Klaus: *Requirements Engineering - Grundlagen, Prinzipien, Techniken*. dpunkt.verlag, 2008.

[PP03] Poppendieck, Mary und Tom Poppendieck: *Lean Software Development: An Agile Toolkit for Software Development Managers*. Addison-Wesley Longman, Mai 2003.

[PPS08] Parsia, Bijan und Peter F. Patel-Schneider: *OWL 2 Web Ontology Language: Primer*. Technischer Bericht, W3C, 2008. `http://www.w3.org/TR/owl2-primer/`.

[PS08] Prud'hommeaux, Eric und Andy Seaborne: *SPARQL Query Language for RDF*, 2008. `http://www.w3.org/TR/rdf-sparql-query/`, W3C Recommendation.

[Res07] RespectIT: *A KAOS Tutorial*, 2007. `http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf`.

[RGS+08a] Röckl, Matthias, Jan Gacnik, Jan Schomerus, Thomas Strang und Matthias Kranz: *Sensing the environment for future driver assistance combining autonomous and cooperative appliances*. In: *Fourth International Workshop on Vehicle-to-Vehicle Communications (V2VCOM)*, Eindhoven, Niederlande, Juni 2008. IEEE ITS Society.

[RGS08b] Röckl, Matthias, Jan Gačnik und Jan Schomerus: *Integration of Car-2-Car Communication as a Virtual Sensor in Automotive Sensor Fusion for Advanced Driver Assistance Systems*. In: *FISITA World Automotive Congress*. VDI/FISITA, September 2008.

[RGW+11] Reschka, Andreas, Jan Gačnik, Jörn Marten Wille, Jürgen Rüdiger Böhmer, Frank Köster und Markus Maurer: *Development of software for open autonomous automotive systems in the stadtpilot-project*. In: *8th International Workshop on Intelligent Transportation (WIT 2011)*, 2011.

[RKS+08] Röckl, Matthias, Frank Korbinian, Thomas Strang, Matthias Kranz, Jan Gacnik und Jan Schomerus: *Hybrid Fusion Approach combining Autonomous and Cooperative Detection and Ranging methods for Situation-aware Driver Assistance Systems*. In: *2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications (CFP08PIM-CDR)*. IEEE, September 2008.

[Roy70] Royce, Winston: *Managing the Development of Large Software Systems*. In: *Proceedings of IEEE WESCON*, Seite 1 bis 9, August 1970.

[RTC92] RTCA - Radio Technical Commission for Aeronautics: *RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, December 1992.

[Sch04] Schwaber, Ken: *Agile Project Management with Scrum*. Microsoft Press, März 2004.

[Sch08] Schnieder, Lars: *Towards terminological rigour in the specification of complex automation systems* . In: *FORMS 2008*, 2008.

[Sch09a] Schneider, Michael: *OWL 2 Web Ontology Language RDF-Based Semantics*. World Wide Web Consortium (W3C), ktoberOktober 2009. `http://www.w3.org/TR/owl2-rdf-based-semantics/`, W3C Recommandation.

[Sch09b] Schnieder, Lars: *Formalisierte Terminologien technischer Systeme und ihrer Zuverlässigkeit*. Dissertation, Technische Universität Braunschweig, 2009.

[SHG+10] Schröder, Mark, Marco Hannibal, Jan Gacnik, Frank Köster, Christian Harms und Tobias Knostmann: *Ein Labor zur modellbasierten Gestaltung interaktiver Assistenz und Automation im Automotive-Umfeld*. In: *AAET 2010*, Braunschweig, Februar 2010. ITS Niedersachsen.

[Sta10] Statistisches Bundesamt: *Verkehrsunfälle 2009 - Unfallentwicklung im Straßenverkehr*, Juli 2010.

[SZ03] Schäuffele, Jörg und Thomas Zurawka: *Automotive Software Engineering*. Vieweg Verlag, 2003.

[TBMM04] Thompson, Henry S., David Beech, Murray Maloney und Noah Mendelsohn: *XML Schema Part 1: Structures*. W3C, 2004. `http://www.w3.org/TR/xmlschema-1/`.

[UNE68] UNECE, United Nations Economic Commission for Europe: *Convention on Road Traffic*, November 1968.

[vL01] Lamsweerde, Axel van: *Building formal requirements models for reliable software*. In: *Ada Europe '01: Proceedings of the 6th Ada-Europe International Conference*

Leuven on Reliable Software Technologies, Seite 1 bis 20, London, UK, 2001. Springer-Verlag, ISBN 3-540-42123-8.

[vOvdLKM00] Ommering, Rob van, Frank van der Linden, Jeff Kramer und Jeff Magee: *The koala component model for consumer electronics software*. Computer, 33(3):78 bis 85, März 2000.

[W3C04] W3C: *OWL-S: Semantic Markup for Web Services*, 2004. `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/`.

[Wei09] Weiser, Martin: *Infotainment Trends 2015 - Welche Chancen haben die OEMs?* Braunschweiger Verkehrskolloquium, April 2009.

[WL99] Weiss, David M. und Chi Tau Robert Lai: *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley Professional, August 1999.

[WNCF89] Wiener, E. L., D. C. Nagel, E. C. Carterette und M. P. Friedman (Herausgeber): *Human Factors in Aviation*. Cognition and Perception. Elsevier, 1989.

[WP08] Wimmel, Harro und Lutz Priese: *Petri-Netze*. Springer-Verlag, 2008.

[WSM10] Wille, Jörn Marten, Falko Saust und Markus Maurer: *Stadtpilot: Driving autonomously on braunschweig's inner ring road*. In: *Intelligent Vehicles Symposium (IV)*, Seiten 506--511. IEEE, 2010.

[YdPLM04] Yu, Yijun, Julio Cesar Sampaio do Prado Leite und John Mylopoulos: *From goals to aspects: Discovering aspects from requirements goal models*. In: *Intl. RE Conf*, Seite 38 bis 47. Society Press, 2004.

[ZVE09] ZVEI - Zentralverband Elektrotechnik und Elektronikindustrie e. V.: *Nationale Roadmap Embedded Systems*, 2009.

[Zwe06] Zwerschke, S.: *Untersuchung zu Bekanntheit, Akzeptanz und Kaufinteresse von Fahrerassistenzsystemen*. In: *VDI-Berichte*, Nummer 1960, 2006.

ISSN 1866-721X